



同济大学交通运输工程学院  
COLLEGE OF TRANSPORTATION ENGINEERING  
TONGJI UNIVERSITY

# 交通数据分析

## 第十、十一讲 神经网络

**沈煜** 博士 副教授

嘉定校区交通运输工程学院311室

yshen@tongji.edu.cn

2022年04月29日

# 计划进度



周	日期	主讲	内容	模块
1	2022.02.25	沈煜	概述	爬虫
2	2022.03.04	沈煜	在线数据采集方法	
3	2022.03.11	沈煜	线性回归模型	
5	2022.03.18	沈煜	广义线性回归	
4	2022.03.25	沈煜	广义线性回归 (作业1)	
6	2022.04.01	沈煜	空间数据描述性分析	
7	2022.04.08	沈煜	空间自回归方法 (作业2)	
8	2022.04.15	沈煜	关联: Apriori	回归分析
9	2022.04.22	沈煜	决策树、支持向量机 (作业3)	
10	2022.04.29	沈煜	浅层神经网络	
11	2022.05.06	沈煜	卷积神经网络 (期末大作业)	
12	2022.05.13	沈煜	经典网络结构	
13	2022.05.20	沈煜	聚类: K-Means、DBSCAN	
14	2022.05.27	沈煜	贝叶斯方法、卡尔曼滤波	
15	2022.06.03	-	端午节放假	机器学习
16	2022.06.10	沈煜	期末汇报 (1)	
17	2022.06.17	沈煜	期末汇报 (2)	

# 主要内容



- 浅层神经网络
- 浅色神经网络的局限
- 卷积神经网络
- LeNet-5手写数字识别



同济大学交通运输工程学院  
COLLEGE OF TRANSPORTATION ENGINEERING  
TONGJI UNIVERSITY

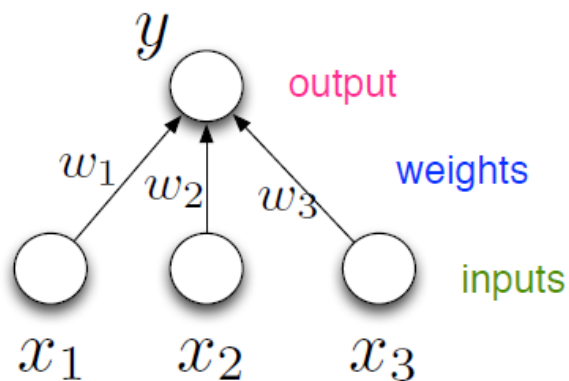
# 浅层神经网络

## 神经元的运算

# 神经网络



- 神经网络模型最早受人脑结构机理启发
- 当今，我们更关注其中的数学与统计学内涵
- 神经网络通过数以千计甚至数以百万计的神经元运算，得到有用的计算结果



$$y = g \left( b + \sum_i x_i w_i \right)$$

Diagram illustrating the mathematical formula for a neuron's output  $y$ . The formula is  $y = g \left( b + \sum_i x_i w_i \right)$ . The components are labeled with colored arrows:  $y$  is the output (pink arrow),  $g$  is the nonlinearity (red arrow),  $b$  is the bias (blue arrow),  $x_i$  is the  $i$ 'th input (green arrow), and  $w_i$  is the  $i$ 'th weight (blue arrow).



# 权重的学习

- $y = x_1 w_1 + x_2 w_2 + x_3 w_3 = w^T X$
- $w = [w_1 \quad w_2 \quad w_3]$
- 首先, 对 $w$ 的值进行随机选取
- 然后, 根据目标值对 $w$ 进行调整
- 使得计算结果趋近于目标

# 权重的学习



➤猜测 $w_1, w_2, w_3$ 等于50

➤计算残差

$$\text{➤ } 850 - (2 \times 50 + 5 \times 50 + 3 \times 50) = 350$$

➤学习规则 (delta-rule) 为:

$$\text{➤ } \Delta w_i = \varepsilon x_i (t - y)$$

➤设学习率为:

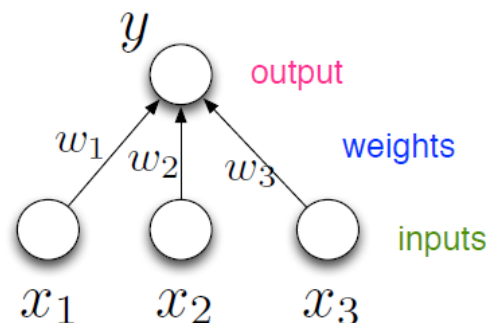
$$\text{➤ } \varepsilon = 1/35$$

➤权重的变化分别为

$$\text{➤ } +20, +50, +30$$

➤新权重调整为: 70, 100, 80

目标值:  $y = 850$



$$y = g \left( b + \sum_i x_i w_i \right)$$

Diagram labels: output (y), bias (b), i'th weight ( $w_i$ ), i'th input ( $x_i$ ), nonlinearity (g).

已知 (样本) :

$$x_1 = 2, x_2 = 5, x_3 = 3$$

# 学习规则 (delta-rule)



➤ 定义误差 (error) 为所有训练样本的残差的平方和

$$➤ E = \frac{1}{2} \sum_{n \in \text{training}} (t^n - y^n)^2$$

➤ 对 $w$ 求导得到误差对于 $w$ 的导数

$$➤ \frac{\partial E}{\partial w_i} = \frac{1}{2} \sum_n \frac{\partial y^n}{\partial w_i} \frac{dE^n}{dy^n} = - \sum_n x_i^n (t^n - y^n)$$

➤ 根据所有训练样本的导数之和, 按比例调整权重

$$➤ \Delta w_i = -\varepsilon \frac{\partial E}{\partial w_i} = \sum_n \varepsilon x_i^n (t^n - y^n)$$



# 学习率 (learning rate)



- 学习率就是梯度下降\上升 (gradient descent\ascent) 法的步长
- 在学习率足够小, 训练时间足够长的情况下, 得到的结果越接近完美值
- 但是, 在实际操作中, 需要 we 根据经验进行选择学习率的大小
  - 学习率太大: 系统不稳定
  - 学习率太小: 学习时间很长

# 逻辑神经元 (Logistic neuron)



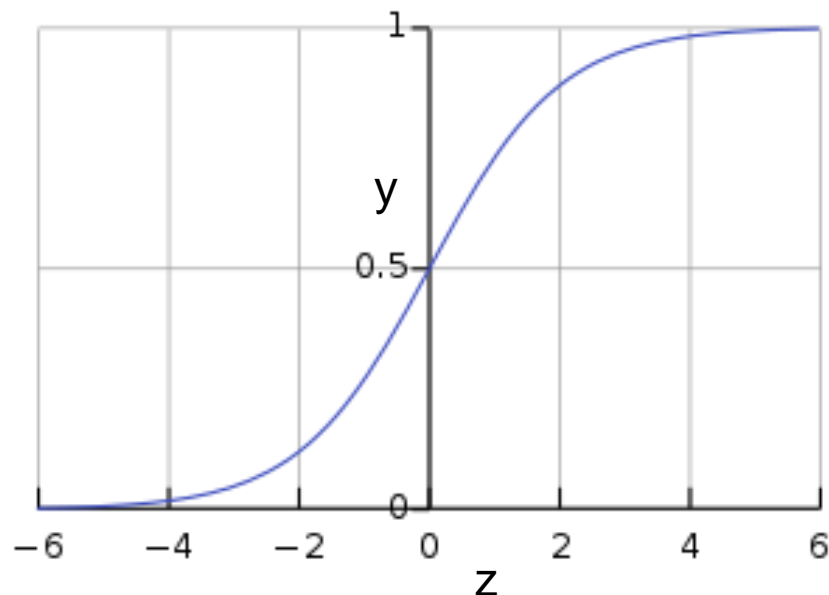
## ➤ 将学习规则扩展到非线性神经元

➤ 逻辑神经元的输出为平滑有边界的实数值，且导数求解容易

$$\text{➤ } z = b + \sum_i x_i w_i$$

$$\text{➤ } y = \frac{1}{1 + e^{-z}}$$

## ➤ 又称为sigmoid函数



# 逻辑神经元的求导



$$\triangleright z = b + \sum_i x_i w_i$$

$$\triangleright \frac{\partial z}{\partial w_i} = x_i$$

$$\triangleright \frac{\partial z}{\partial x_i} = w_i$$

$$\triangleright \frac{\partial y}{\partial w_i} = \frac{\partial z}{\partial w_i} \frac{dy}{dz} = x_i y(1 - y)$$

$$\triangleright \frac{\partial E}{\partial w_i} = \sum_n \frac{\partial y^n}{\partial w_i} \frac{\partial E}{\partial y^n}$$

$$\triangleright = -\sum_n x_i^n y^n (1 - y^n) (t^n - y^n)$$

学习规则

$$\triangleright y = \frac{1}{1+e^{-z}}$$

$$\triangleright \frac{dy}{dz} = -\left(\frac{1}{1+e^{-z}}\right)^2 e^{-z}(-1)$$

$$\triangleright = \frac{1+e^{-z}-1}{(1+e^{-z})^2}$$

$$\triangleright = \frac{1}{1+e^{-z}} \frac{1+e^{-z}-1}{1+e^{-z}}$$

$$\triangleright = \frac{1}{1+e^{-z}} \left(1 - \frac{1}{1+e^{-z}}\right)$$

$$\triangleright = y(1 - y)$$

# 常用的激活函数



## ➤ sigmoid函数

$$\text{➤ } y = \frac{1}{1+e^{-z}}, \quad \frac{dy}{dz} = y(1-y)$$

## ➤ tanh函数

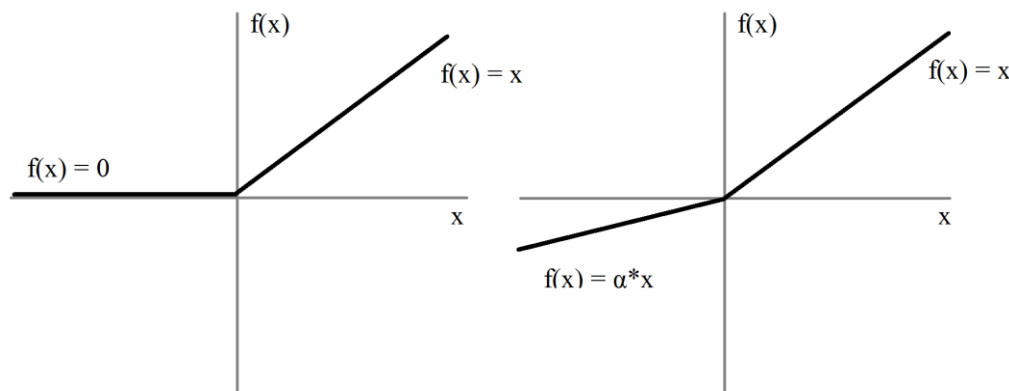
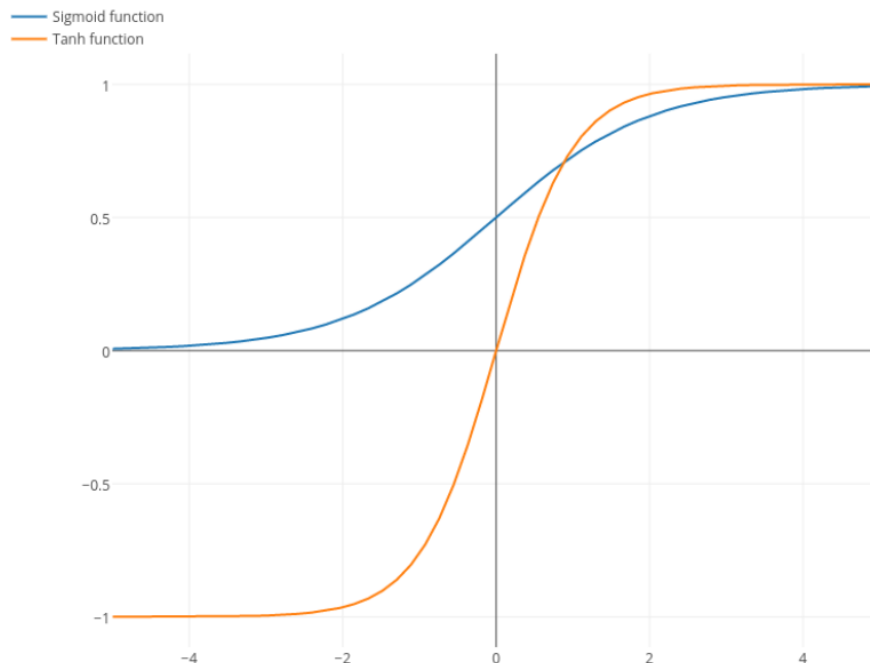
$$\text{➤ } y = \frac{e^z - e^{-z}}{e^z + e^{-z}}, \quad \frac{dy}{dz} = 1 - y^2$$

## ➤ ReLU函数

$$\text{➤ } y = \max(0, z)$$

## ➤ Leaky ReLU函数

$$\text{➤ } y = \max(0.01z, z)$$

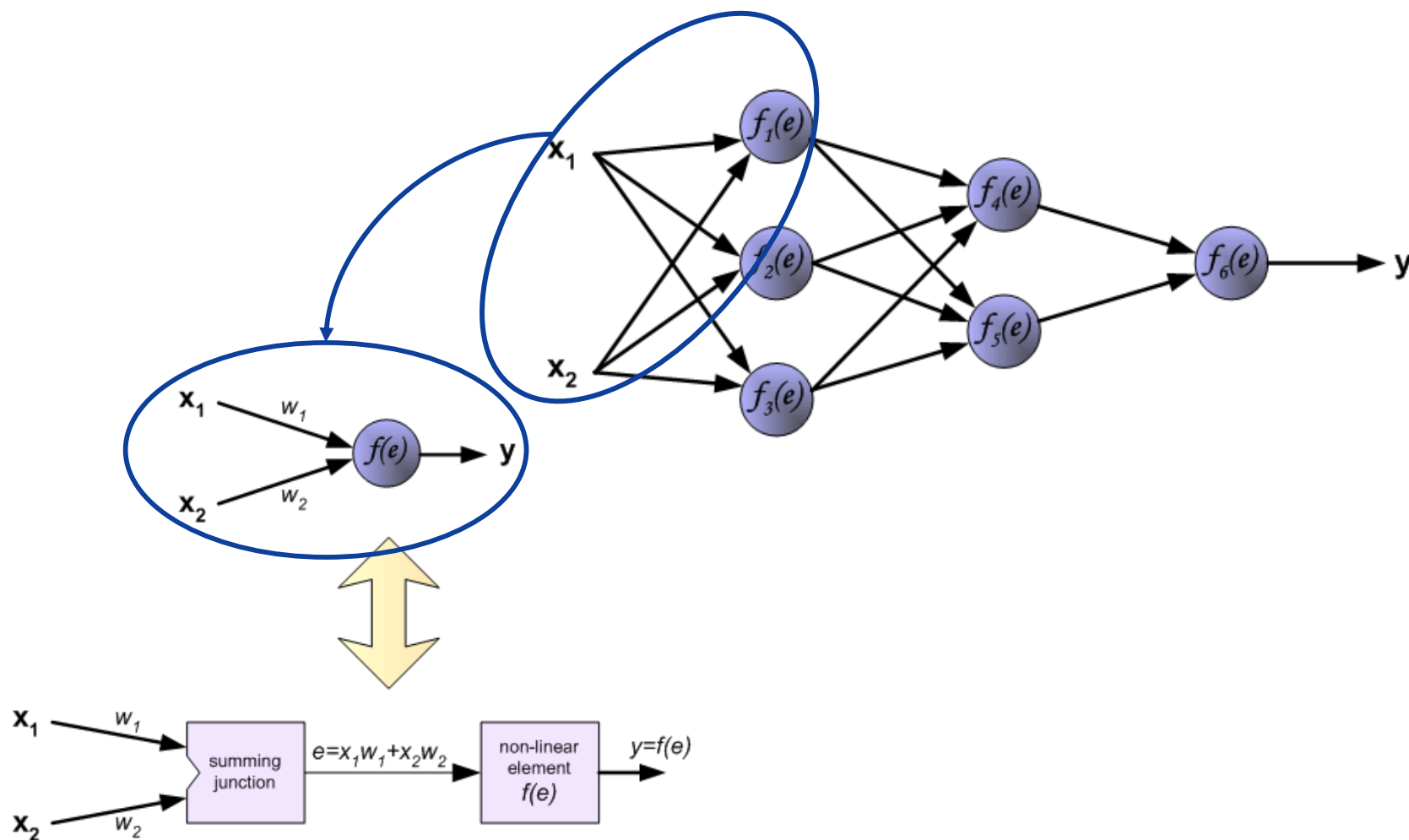


# 四种激活函数的比较

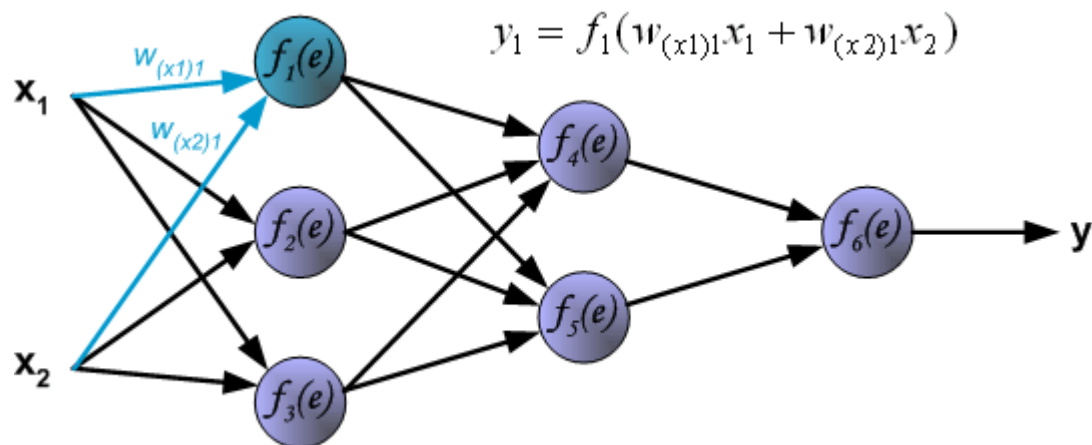


- 相比sigmoid, tanh的表现效果更好
- sigmoid用在二元分类的输出层
- 默认激活函数是ReLU
- Leaky ReLU的效果比ReLU更好, 但没有ReLU常用
- 使用ReLU或Leaky ReLU时
  - 神经网络的训练速度比tanh或sigmoid快很多
- sigmoid和tanh的缺点在于
  - 当 $z$ 特别大或特别小时, 函数的斜率会很小

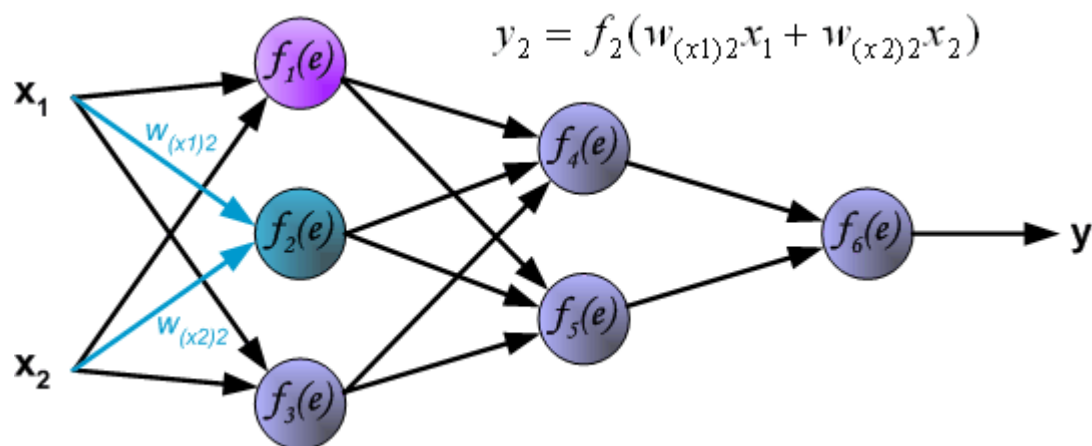
# 神经网络的传递



# 正向传播：第一层

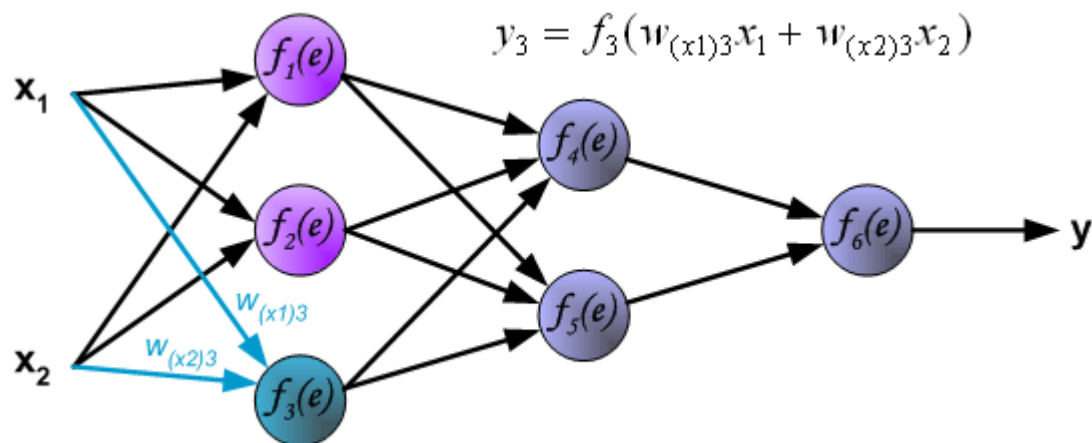


# 正向传播：第一层

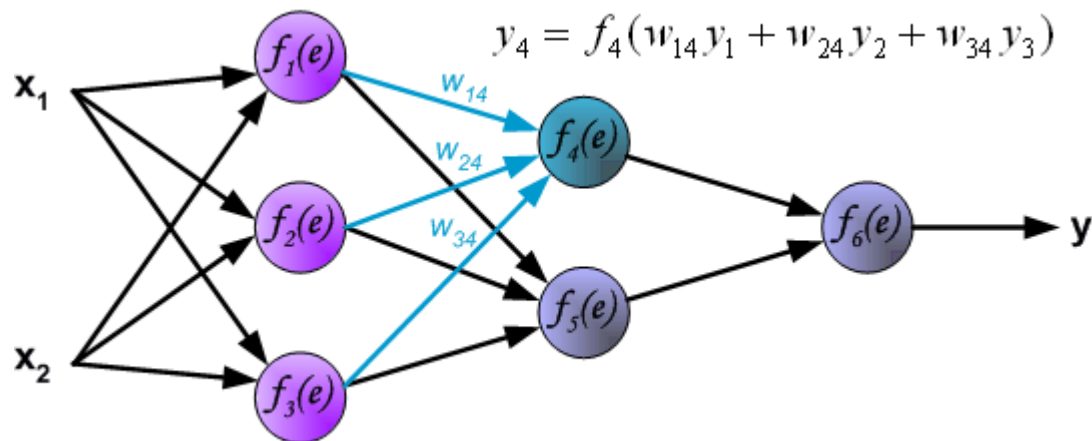




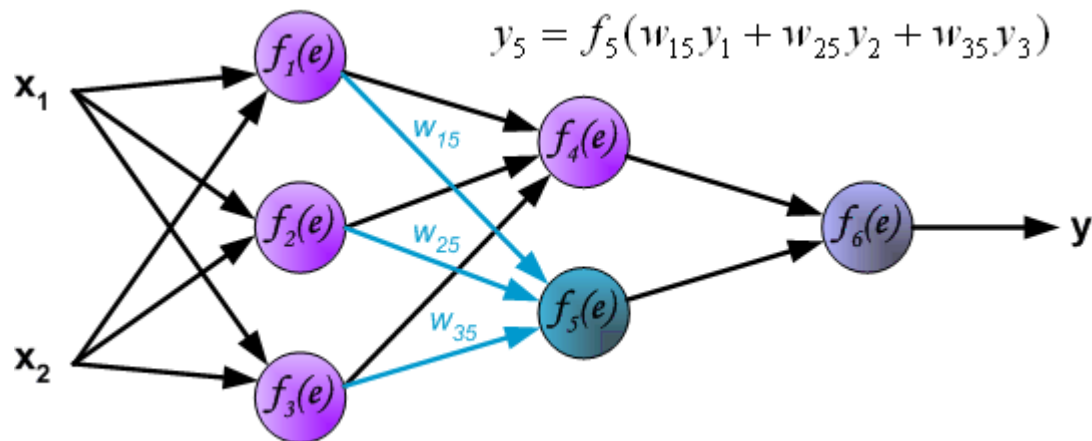
# 正向传播：第一层



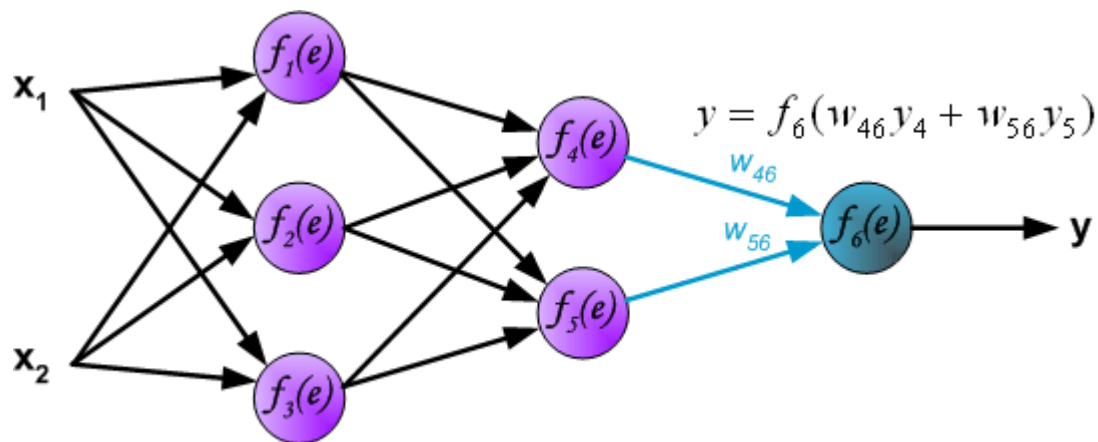
# 正向传播：第二层



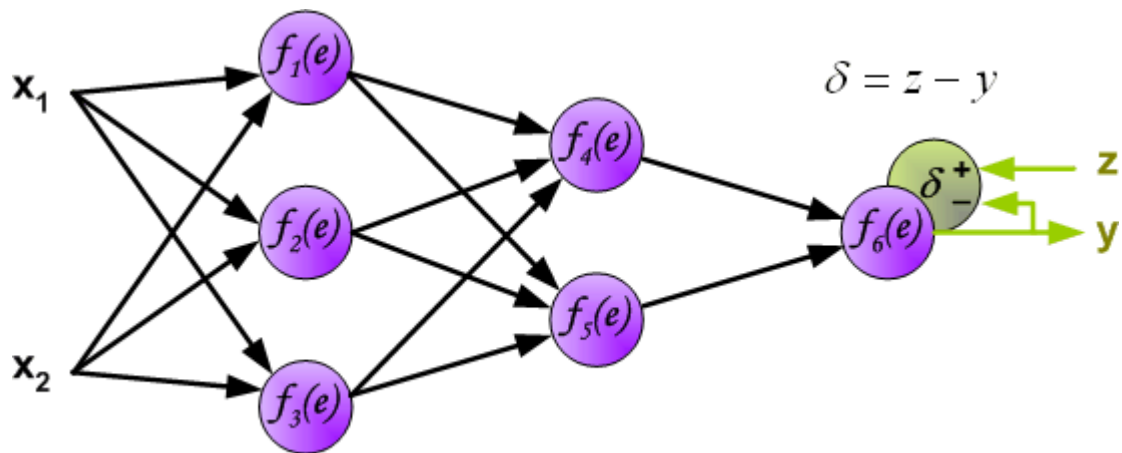
# 正向传播：第二层



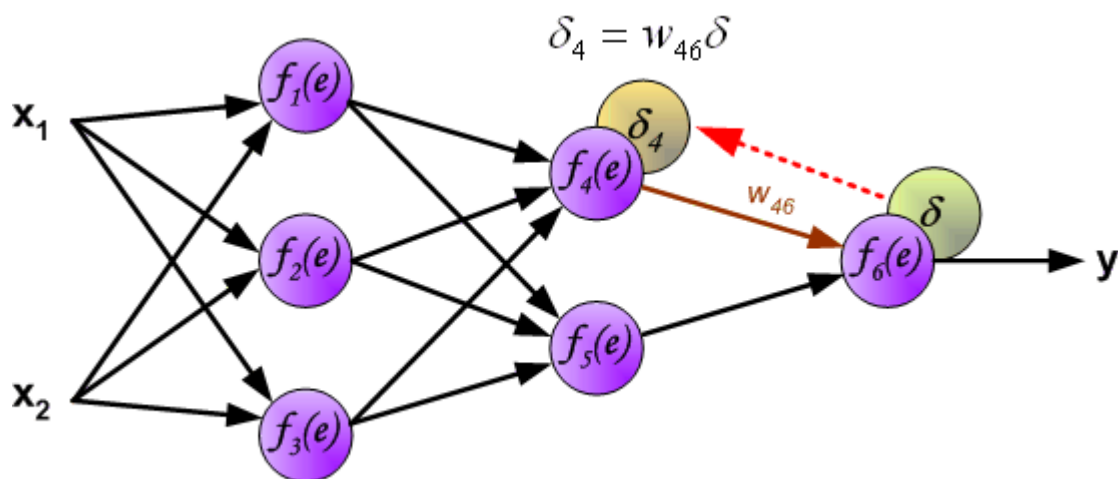
# 正向传播：输出层



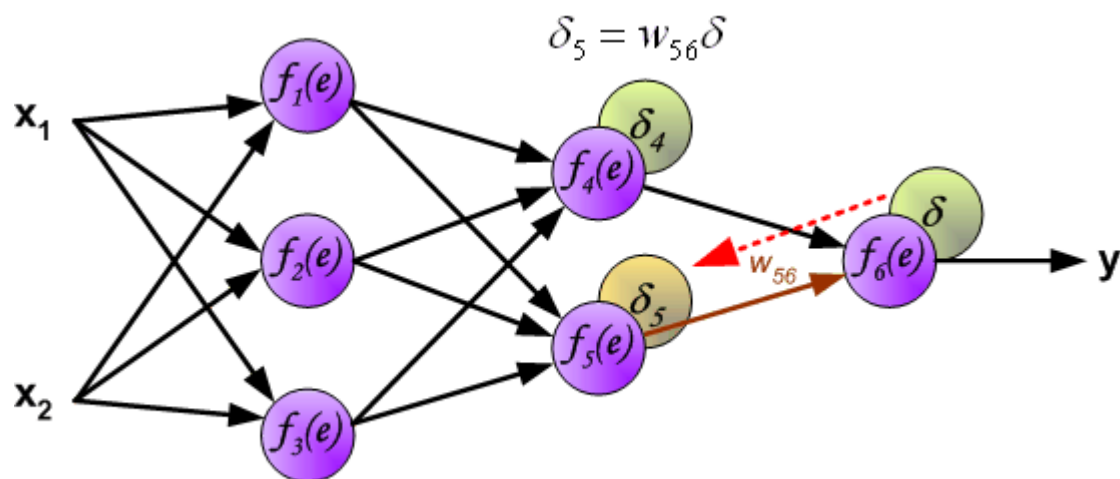
# 反向传播：输出层

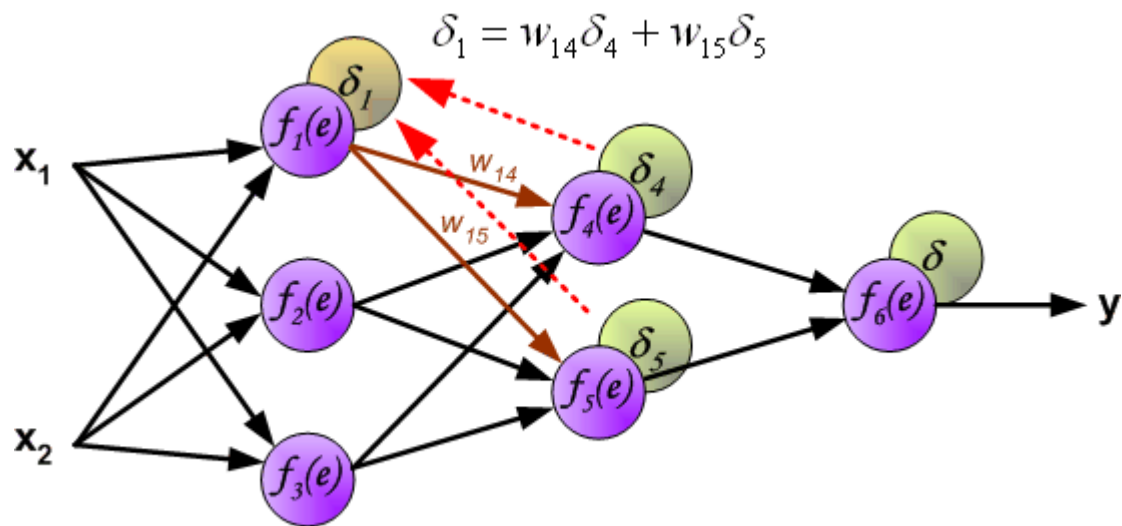


# 反向传播：第二层



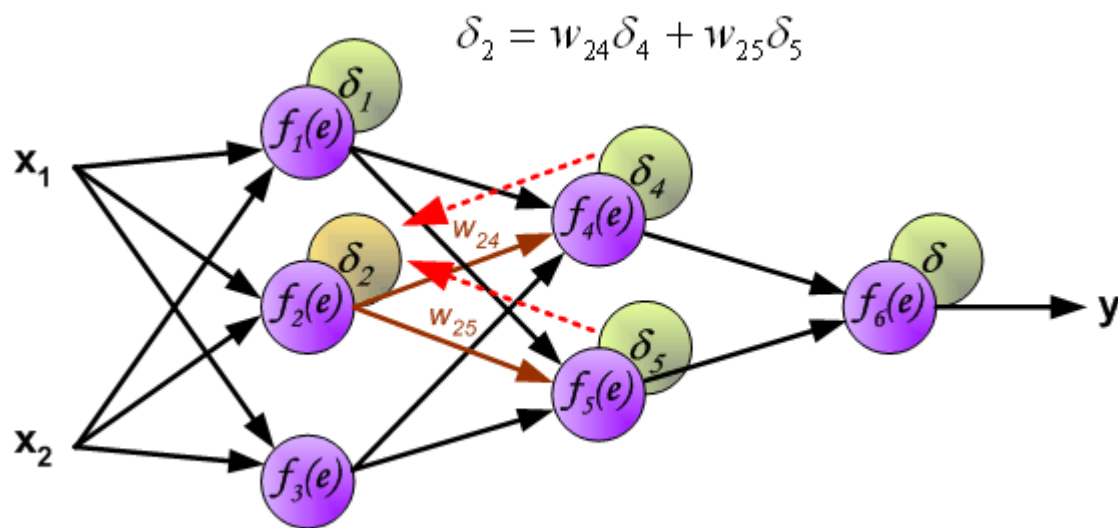
# 反向传播：第二层



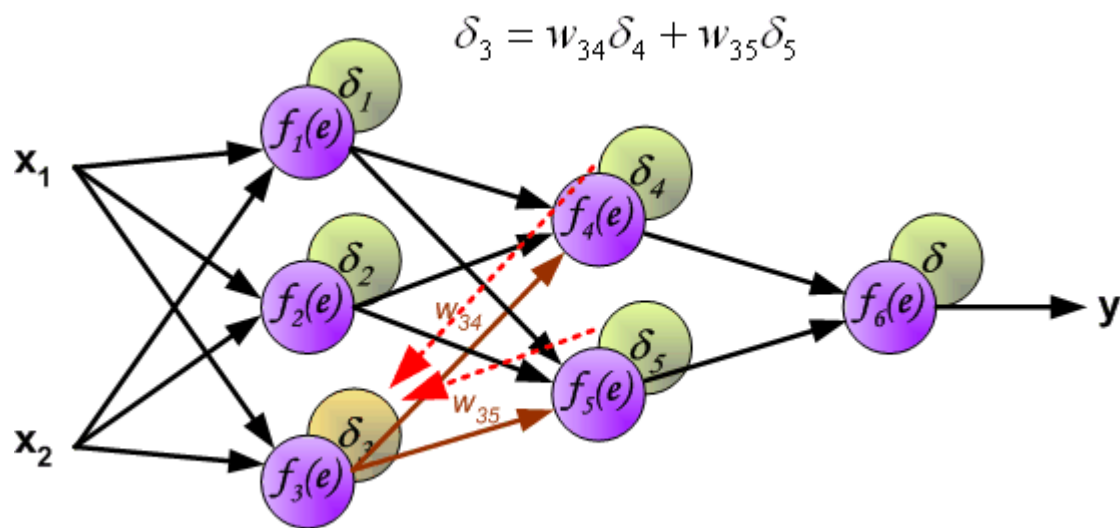




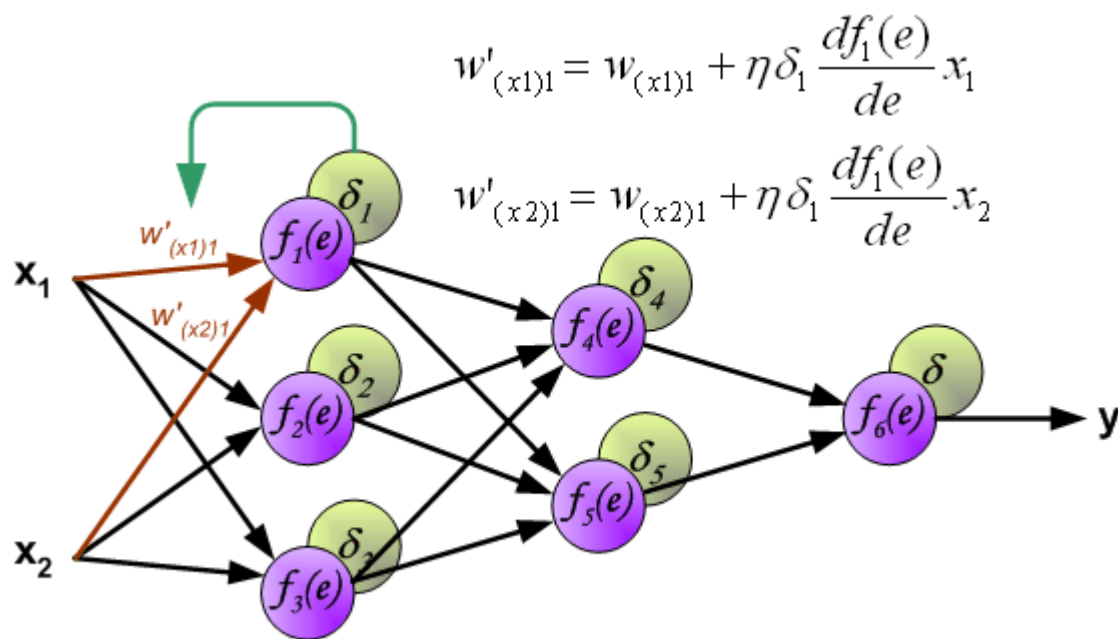
# 反向传播：第一层



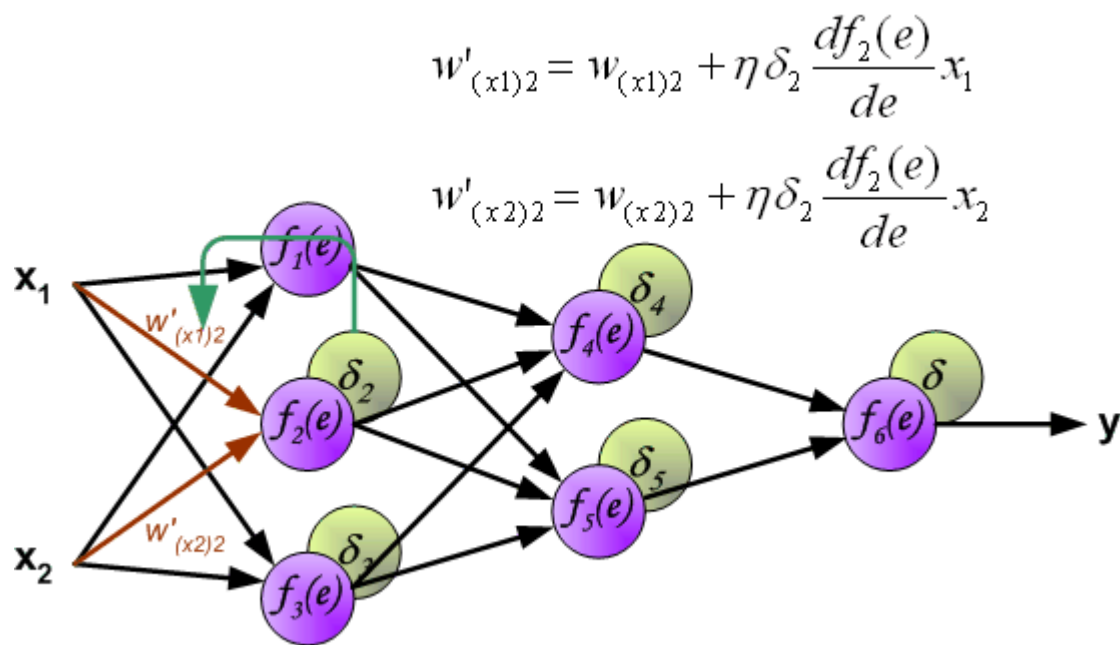
# 反向传播：第一层



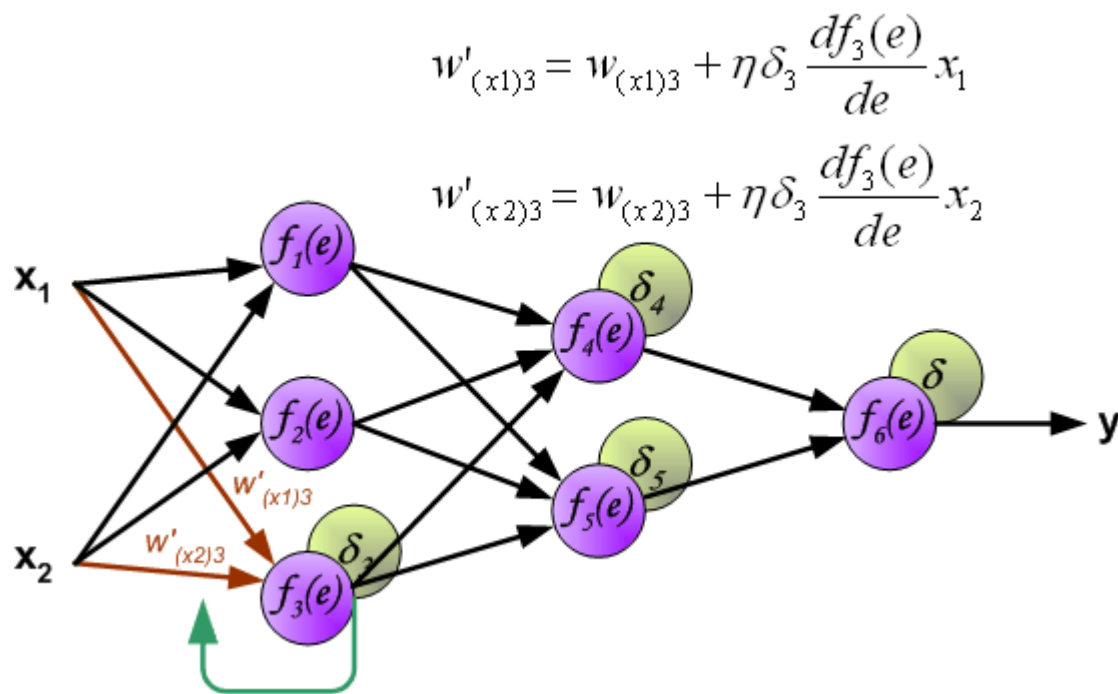
# 调整权重，向前传播：第一层



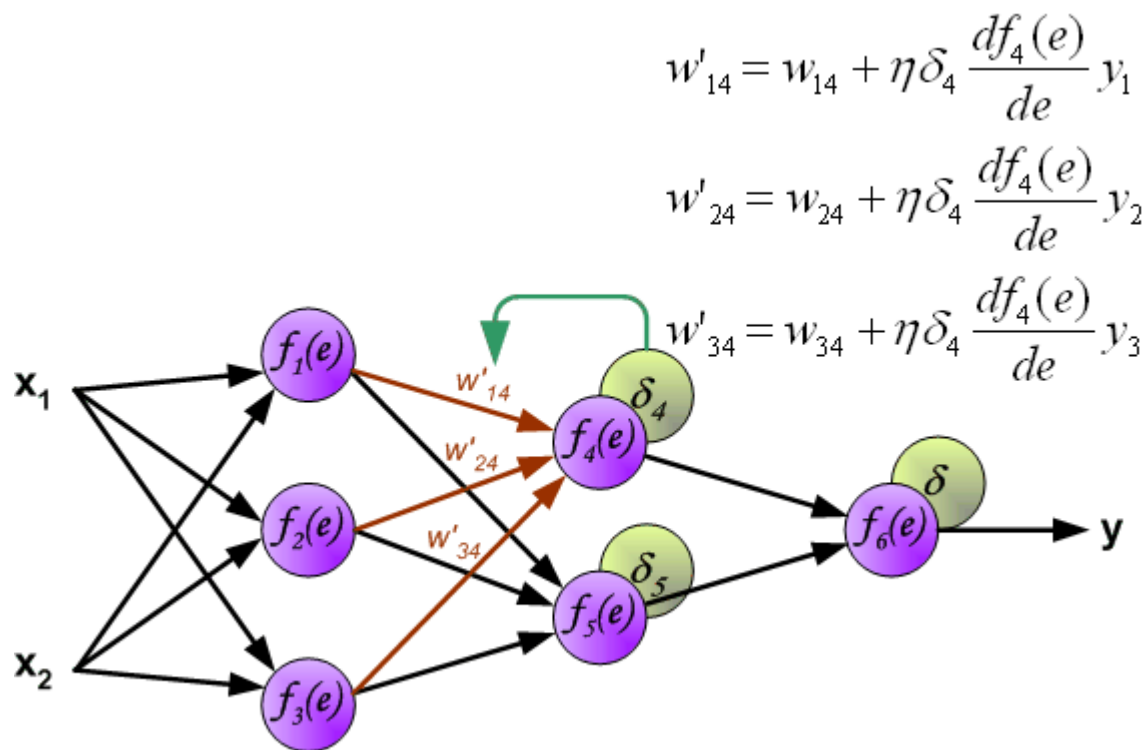
# 调整权重，向前传播：第一层



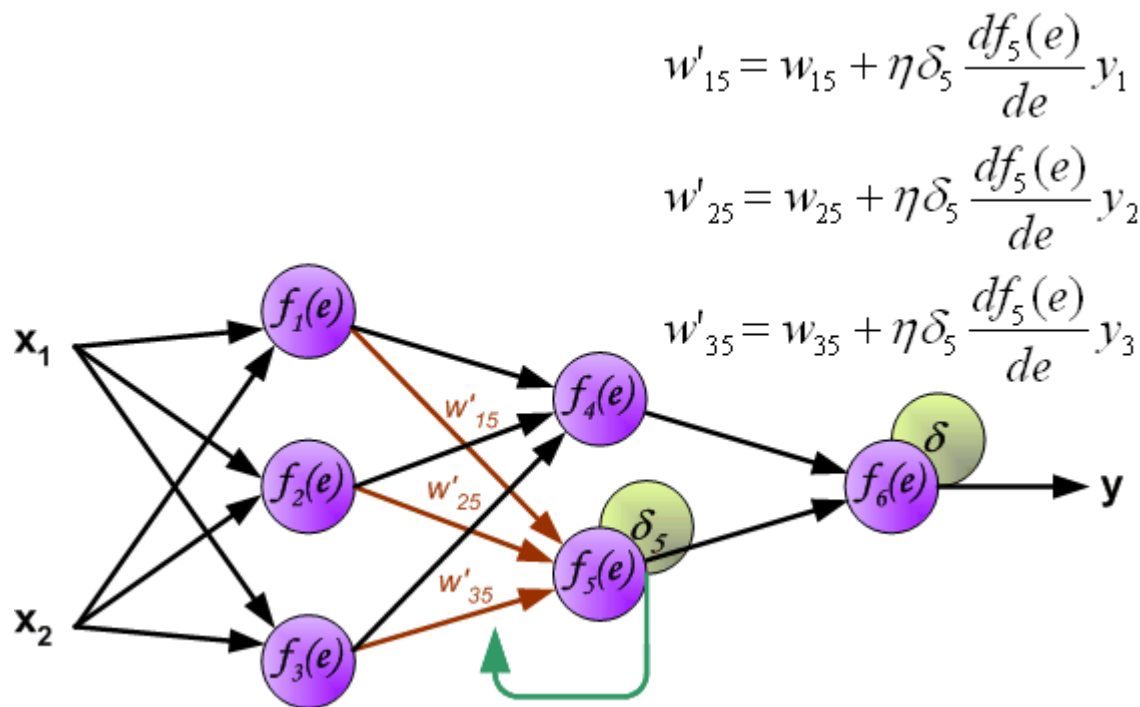
# 调整权重，向前传播：第一层



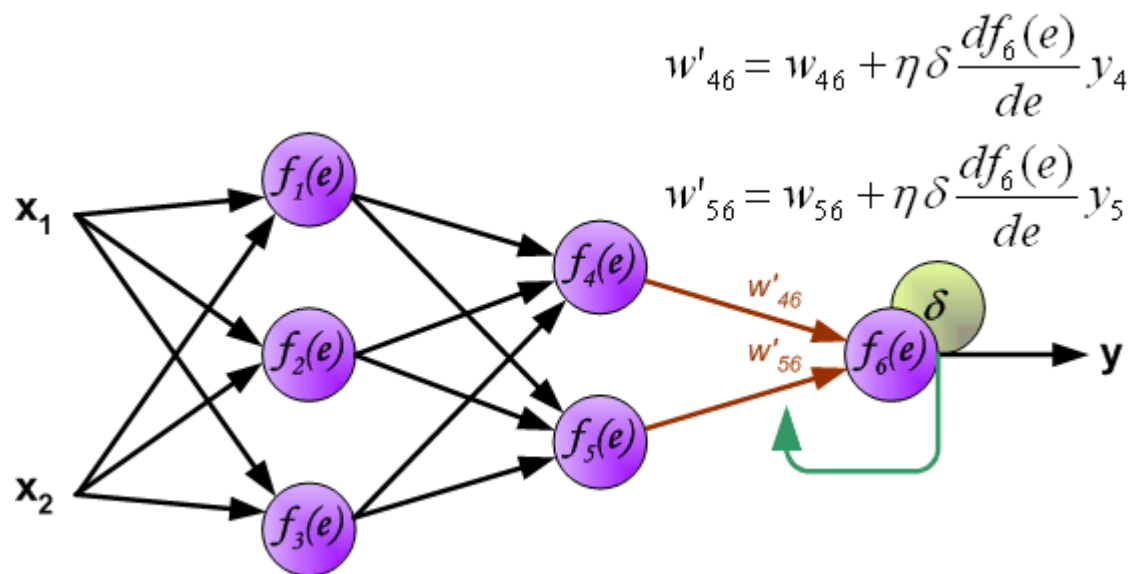
# 调整权重，向前传播：第二层



# 调整权重，向前传播：第二层



# 调整权重，向前传播：输出层





# 参数的初始化



## ➤ 初始化为零

$$\begin{aligned} \text{➤ } W_1 &= \begin{bmatrix} w_{14} & w_{24} & w_{34} \\ w_{15} & w_{25} & w_{35} \end{bmatrix} = \\ &\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \end{aligned}$$

$$\text{➤ } W_2 = [w_{46} \quad w_{56}] = [0 \quad 0]$$

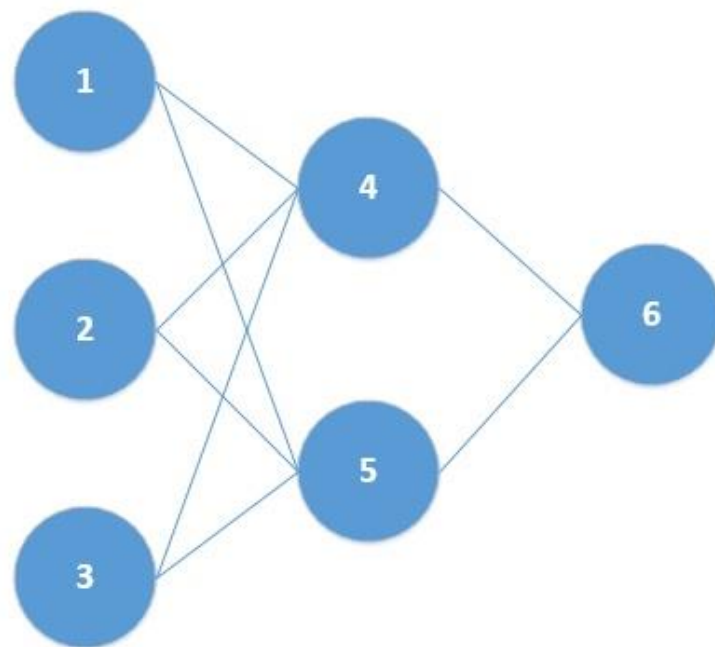
$$\text{➤ } \begin{bmatrix} z_4 \\ z_5 \end{bmatrix} =$$

$$\begin{bmatrix} w_{14} & w_{24} & w_{34} \\ w_{15} & w_{25} & w_{35} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

$$\text{➤ } z_4 = z_5 = 0$$

$$\text{➤ } a_4 = a_5 = f(z_4) = f(z_5)$$

$$\text{➤ } z_6 = [w_{46} \quad w_{56}] \begin{bmatrix} x_4 \\ x_5 \end{bmatrix} = 0$$



# 参数的初始化



- 通过反向传播，结点4、5的梯度改变是一样的，设为 $\Delta w$
- 此时：
  - $w_{46} = 0 + \Delta w = \Delta w$
  - $w_{56} = 0 + \Delta w = \Delta w$
  - $w_{46} = w_{56}$ ，新的参数相同
- 同理，更新之后， $w_{14}, \dots, w_{35}$ 也是相同的
- 无论进行多少次正向与反向传播，两层之间的参数以及输出都是一样的
- 后果：不同结点无法学习到不同的特征，每层相当于只有一个结点
- 解决方案：将 $w$ 初始化为随机值



同济大学交通运输工程学院  
COLLEGE OF TRANSPORTATION ENGINEERING  
TONGJI UNIVERSITY

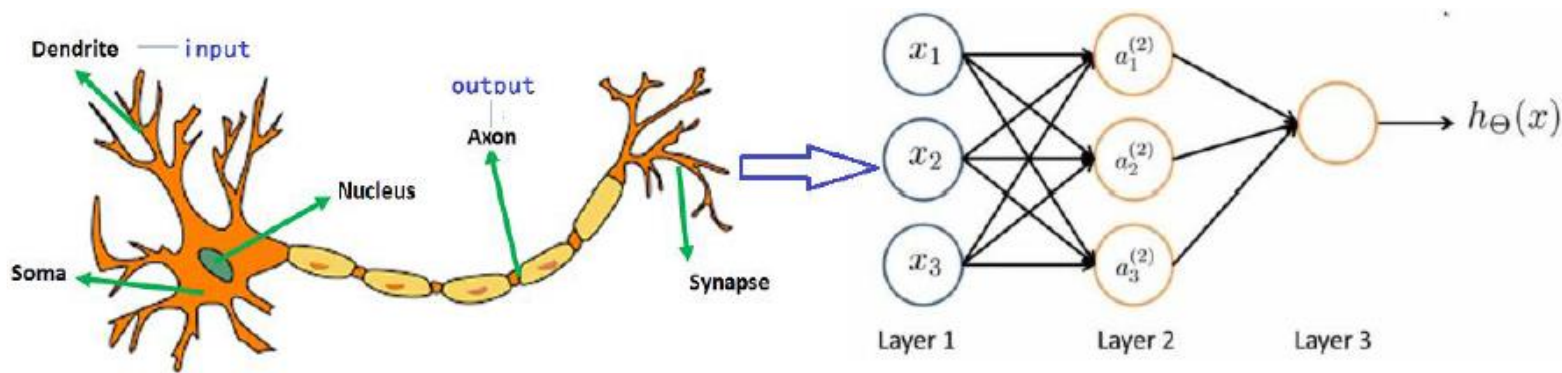
# 浅层神经网络的局限

# 总结：浅层神经网络



- 卷积神经网络，是在神经网络的基础上进化而来的。
- 神经网络的本质是一个分类器，如果将一个网络视为黑箱，那么它的输入是一个对象的特征向量，输出是这个对象所属的类别。
- 主要由三个部分组成：
  - 神经元 (Cell)
  - 损失函数 (loss function)
  - 激活函数 (activation function)

# 神经元



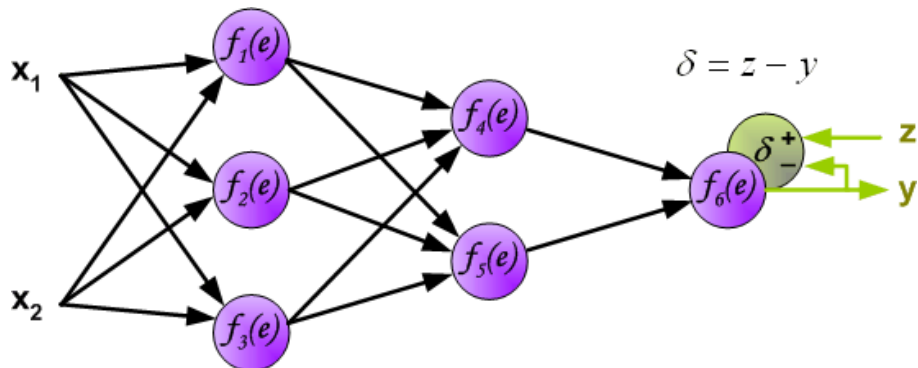
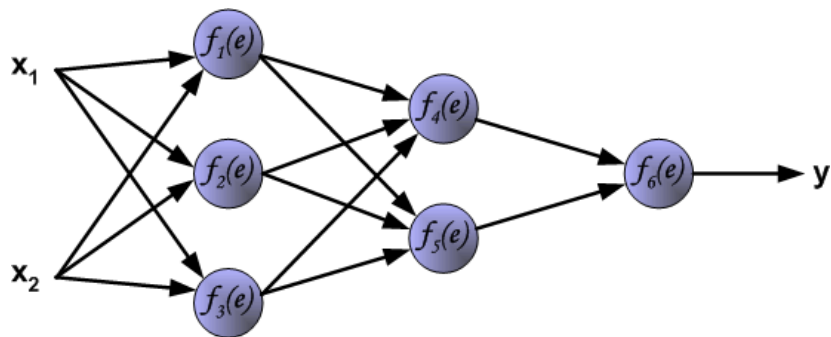
- Layer1: 输入层，众多神经元接受大量非线性输入消息。
- Layer2: 隐藏层，是输入层和输出层之间众多神经元和链接组成的各个层面。隐层可以有一层或多层。隐层的节点（神经元）数目不定，但数目越多神经网络的非线性越显著，从而神经网络的鲁棒性更显著。
- Layer3: 输出层，消息在神经元链接中传输、分析、权衡，形成输出结果。

# 损失函数(Loss Function)



- 一个网络的初始参数是完全随机的，这样将数据输入网络，得到的输出也是完全随机的。
- 但是在训练过程中，输入的数据是有标注的，即我们知道每一条输入数据的真实类别。
- 在每一次训练后，损失函数会获得本次的分类结果，通过将每一条结果与真实类别比较，损失函数会给出本次分类与真实类别的差距。这个差距将指导神经网络更新参数。

# 损失函数(Loss Function)



➤ 损失函数具体的实现形式有很多

➤ 包括softmax loss、SVM loss等等。

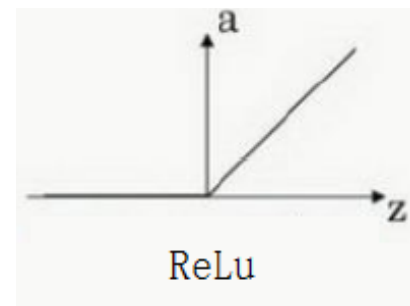
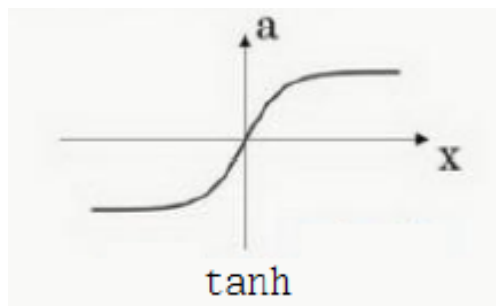
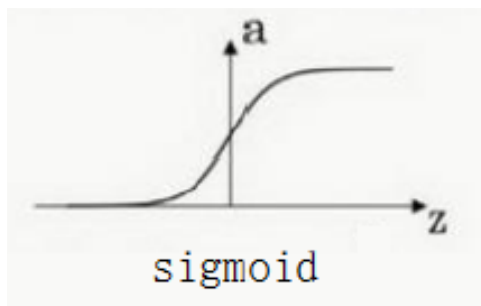
➤ 损失函数的作用是衡量一次分类结果与真实类别的差距

➤ 如果一次分类全部正确，那么那次的损失 (loss) 就是0

# 激活函数(Activation Function)



- 因为线性模型的表达能力不够，所以，为了让神经网络计算出有效的函数，必须使用非线性激活函数，加入非线性因素。
- 例如：





➤神经网络曾经是机器学习领域非常热门的方向

➤缺陷：

➤难以进行理论分析

➤训练方法需要很多经验和技巧

➤巨大的计算量和优化求解难度

- 误差反向传播算法 (Back Propagation, BP算法)
  - 通过梯度下降方法在训练过程中修正权重使得网络误差最小
  - 在层次深的情况下性能变得很不理想
  - 传播时容易出现所谓的梯度弥散Gradient Diffusion或称之为梯度消失,
- 非凸目标代价函数导致求解陷入局部最优
  - 这种情况随着网络层数的增加而更加严重
  - 随着梯度的逐层不断消散导致其对网络权重调整的作用越来越小
- 只能转而处理浅层结构, 从而限制了性能
  - 小于等于3

- 在有限样本和计算单元的情况下对复杂函数的表示能力有限
- 针对复杂分类问题的泛化能力受到一定的制约
- 需要依靠人工来抽取样本的特征
  - 手工地选取特征非常费力
  - 很大程度上靠经验和运气

# 计算机视觉(Computer Vision)



- 处理一张 $1000 \times 1000$  (约1M) 的图片
- 特征向量的维度达到了 $1000 \times 1000 \times 3$  (RGB通道) ,  
即特征为300万
- 假设采用全连接层 (FCN) , 第一个隐藏层中有  
1000个隐藏单元, 则 $W^{[1]}$ 会有30亿个参数



同济大学交通运输工程学院  
COLLEGE OF TRANSPORTATION ENGINEERING  
TONGJI UNIVERSITY

# 卷积神经网络

➤ CNN, 即convolutional neural network, 卷积神经网络, 是近些年来在计算机视觉领域里程碑式的算法。在目标检测、对象识别、图像分割、超分辨、对象跟踪、对象检索等等一些列子问题中, 基于CNN算法的识别精度都远远高于传统算法, 可以说CNN正是推动此次人工智能浪潮的主要力量。

- CNN的结构受到著名的Hubel-Wiesel生物视觉模型的启发，尤其是模拟视觉皮层V1和V2层中Simple Cell和Complex Cell的行为。
  - 1959年，Hubel & Wiesel发现，动物视觉皮层细胞负责检测光学信号。
- 受此启发，1980年 Kunihiro Fukushima 提出了CNN的前身——neocognitron
- 1989年，Yann LeCun等人发表论文，确立了CNN的现代结构，后来又对其进行完善。他们设计了一种多层的人工神经网络，取名叫做LeNet-5，可以对手写数字做分类。

- K. Fukushima, “Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position,” *Biological Cybernetics*, vol. 36, pp. 193–202, 1980
- Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, “Backpropagation applied to handwritten zip code recognition,” *Neural Computation*, vol. 1, no. 4, pp. 541–551, 1989
- Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998



# 卷积的数学表达



## ➤积分形式

➤  $s(t) = \int x(a)w(t-a)da$

➤ 常用表达式  $s(t) = (x * w)(t)$

## ➤离散形式

➤ 一维情况  $s[t] = (x * w)(t) = \sum_{a=-\infty}^{+\infty} x[a]w(t-a)$

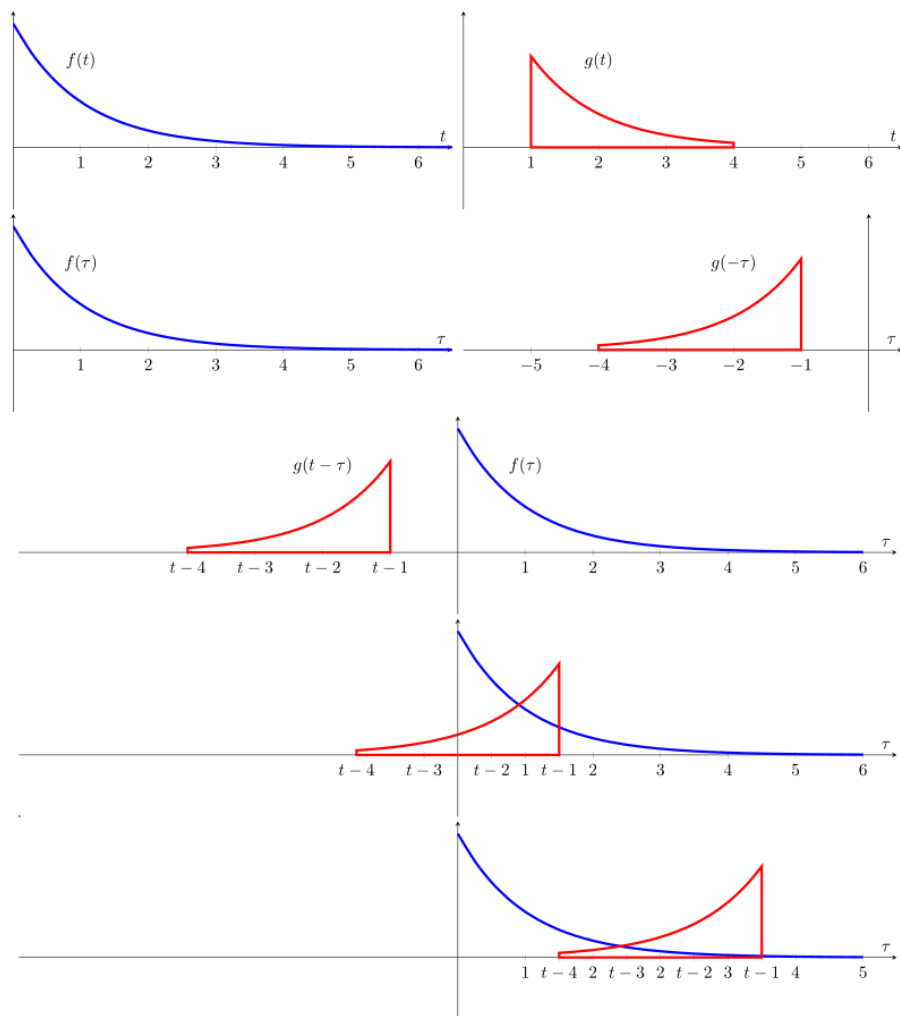
### ➤二维情况

➤  $s[i, j] = (I * K)[i, j] = \sum_m \sum_n I[m, n]K[i-m, j-n]$

➤  $s[i, j] = (I * K)[i, j] = \sum_m \sum_n I[i-m, j-n]K[m, n]$

## ➤K称为kernel

# 卷积



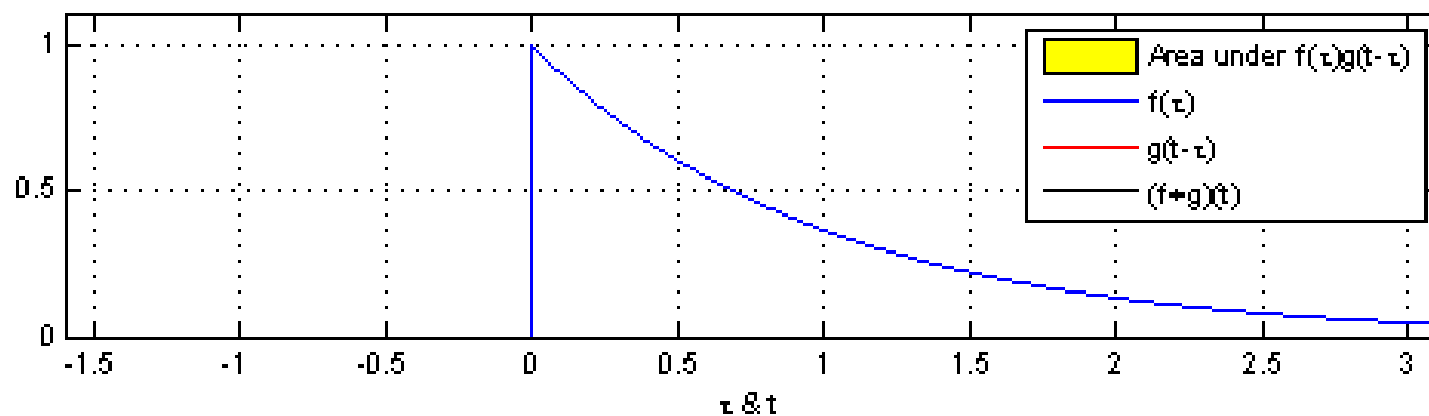
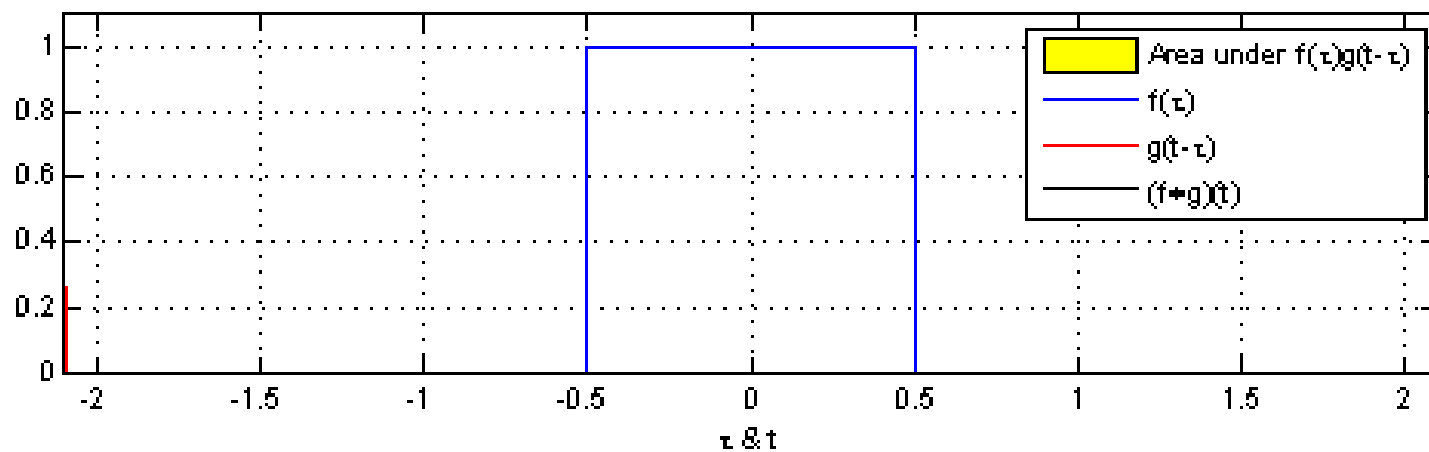
➤通过两个函数 $f$ 和 $g$ 生成第三个函数的一种数学算子

➤表征函数 $f$ 与经过翻转和平移的 $g$ 的乘积函数所围成的曲边梯形的面积

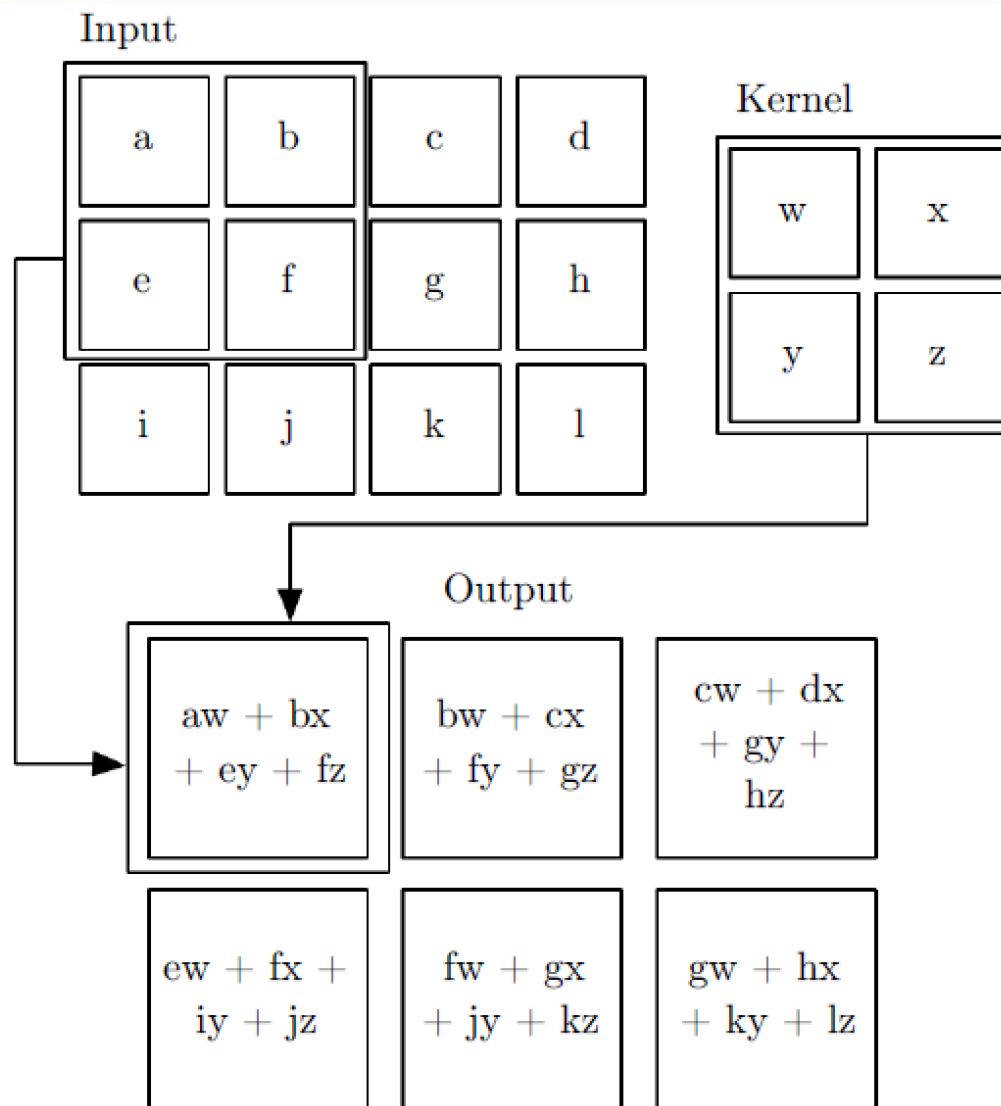
➤如果将参加卷积的一个函数看作区间的指示函数

➤卷积还可以被看作是“移动平均”的推广

# 卷积的过程



# CNN中的卷积



# CNN中的卷积



➤ 设卷积核为

1	0	1
0	1	0
1	0	1

➤ 对图片进行卷积:

1 <sub>x1</sub>	1 <sub>x0</sub>	1 <sub>x1</sub>	0	0
0 <sub>x0</sub>	1 <sub>x1</sub>	1 <sub>x0</sub>	1	0
0 <sub>x1</sub>	0 <sub>x0</sub>	1 <sub>x1</sub>	1	1
0	0	1	1	0
0	1	1	0	0

Image

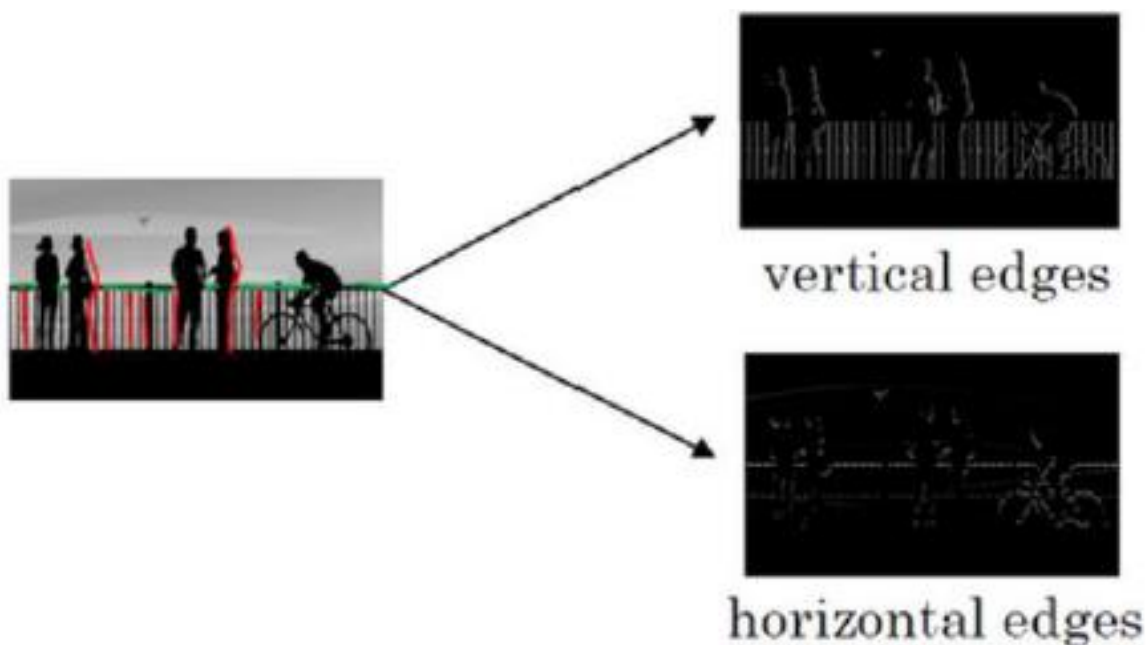
4		

Convolved  
Feature

# 边缘检测



- 给出这样一张图，让电脑“看清楚”里面是什么物体。
- 做的第一件事是：检测图片中的边缘



# 卷积运算进行边缘检测



➤对  $6 \times 6 \times 1$  灰度图像，检测垂直边缘

➤构造矩阵，即“过滤器” (filter)

➤构造的垂直边缘的过滤器：
$$\begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}$$

➤对  $6 \times 6$  的图像进行卷积运算

3	0	1	2	7	4
1	5	8	9	3	1
2	7	2	5	1	3
0	1	3	1	7	8
4	2	1	6	2	8
2	4	5	2	3	9

\*

1	0	-1
1	0	-1
1	0	-1

# 卷积运算进行边缘检测



➤输出将会是一个 $4 \times 4$ 矩阵，也可以看做是一个 $4 \times 4$ 的图像

3	0	1	2	7	4
1	5	8	9	3	1
2	7	2	5	1	3
0	1	3	1	7	8
4	2	1	6	2	8
2	4	5	2	3	9

\*

1	0	-1
1	0	-1
1	0	-1

=

$a_1$			

$$a_1 = 3 \times 1 + 1 \times 1 + 2 \times 1 + 0 \times 0 + 5 \times 0 + 7 \times 0 + 1 \times (-1) + 8 \times (-1) + 2 \times (-1) = -5$$



# 右移一步



3	0	1	2	7	4
1	5	8	9	3	1
2	7	2	5	1	3
0	1	3	1	7	8
4	2	1	6	2	8
2	4	5	2	3	9

\*

1	0	-1
1	0	-1
1	0	-1

=

-5	-4		

3	0	1	2	7	4
1	5	8	9	3	1
2	7	2	5	1	3
0	1	3	1	7	8
4	2	1	6	2	8
2	4	5	2	3	9

\*

1	0	-1
1	0	-1
1	0	-1

=

-5	-4	0	

# 继续移动



3	0	1	2	7	4
1	5	8	9	3	1
2	7	2	5	1	3
0	1	3	1	7	8
4	2	1	6	2	8
2	4	5	2	3	9

\*

1	0	-1
1	0	-1
1	0	-1

=

-5	-4	0	8
-10	-2	2	3
0	-2	-4	-7
-3	-2	-3	-16

# 垂直边缘检测



10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0



\*

1	0	-1
1	0	-1
1	0	-1



=

0	30	30	0
0	30	30	0
0	30	30	0
0	30	30	0



# 卷积运算



- 过滤器不只是前面一种数字组合，也有其他组合
- 把过滤器的9个数字当作9个参数，使用BP算法，让神经网络自动去学习它们。

3	0	1	2	7	4
1	5	8	9	3	1
2	7	2	5	1	3
0	1	3	1	7	8
4	2	1	6	2	8
2	4	5	2	3	9

\*

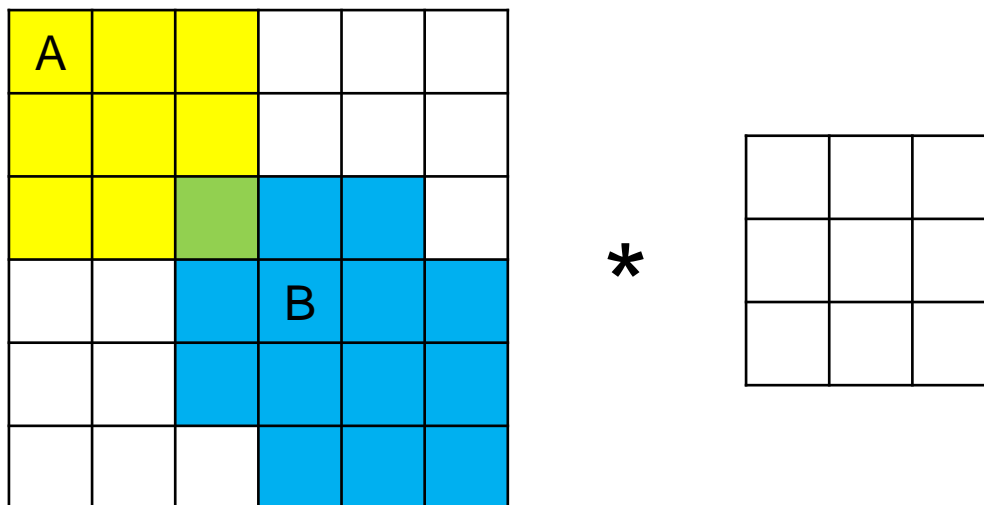
$w_1$	$w_2$	$w_3$
$w_4$	$w_5$	$w_6$
$w_7$	$w_8$	$w_9$

=


# 卷积运算：填充Padding



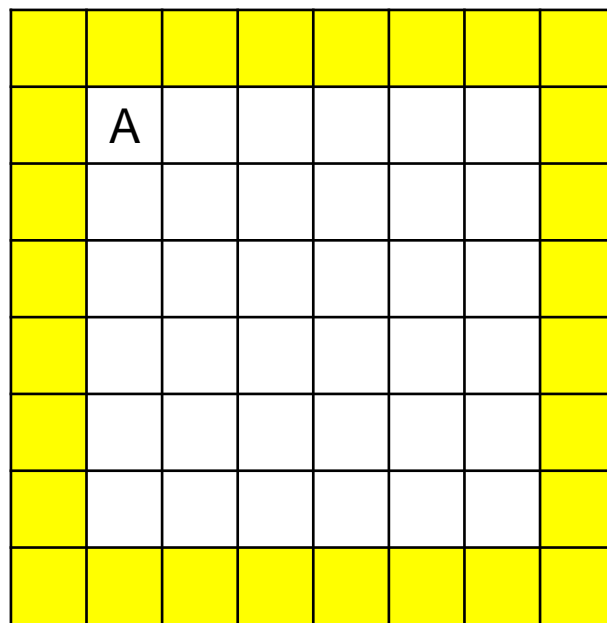
- 使用 $3 \times 3$ 的过滤器，会有两个缺点
  - 每做一次卷积运算，图像会缩小
  - 角落边缘的像素，只被一个输出使用
    - 如：A位置只能使用一次；B位置可以使用多次



# 卷积运算：填充Padding



- 解决方法：做卷积之前，填充这幅图像
  - 卷积之后，得到了一个和原始图像相同大小的图像
  - A位置也有重叠，角落的信息利用提高了
- 两种填充方法
  - Valid：不填充
  - Same：输出图像大小 = 原始图像大小



# 卷积运算：步长Stride



➤ Stride=2

2	3	7	4	6	2	9
6	6	9	8	7	4	3
3	4	8	3	8	9	7
7	8	3	6	6	3	4
4	2	1	8	3	4	6
3	2	4	1	9	8	3
0	1	3	9	2	1	4

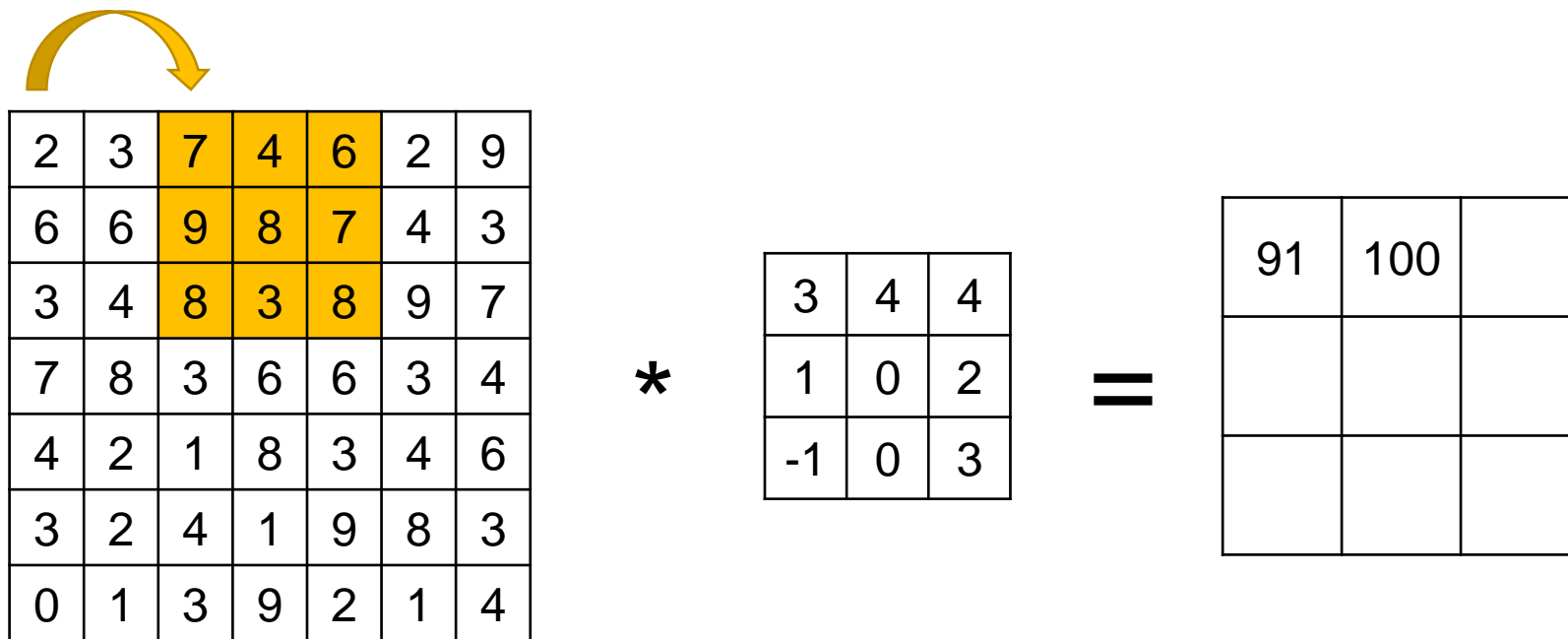
\*

3	4	4
1	0	2
-1	0	3

=

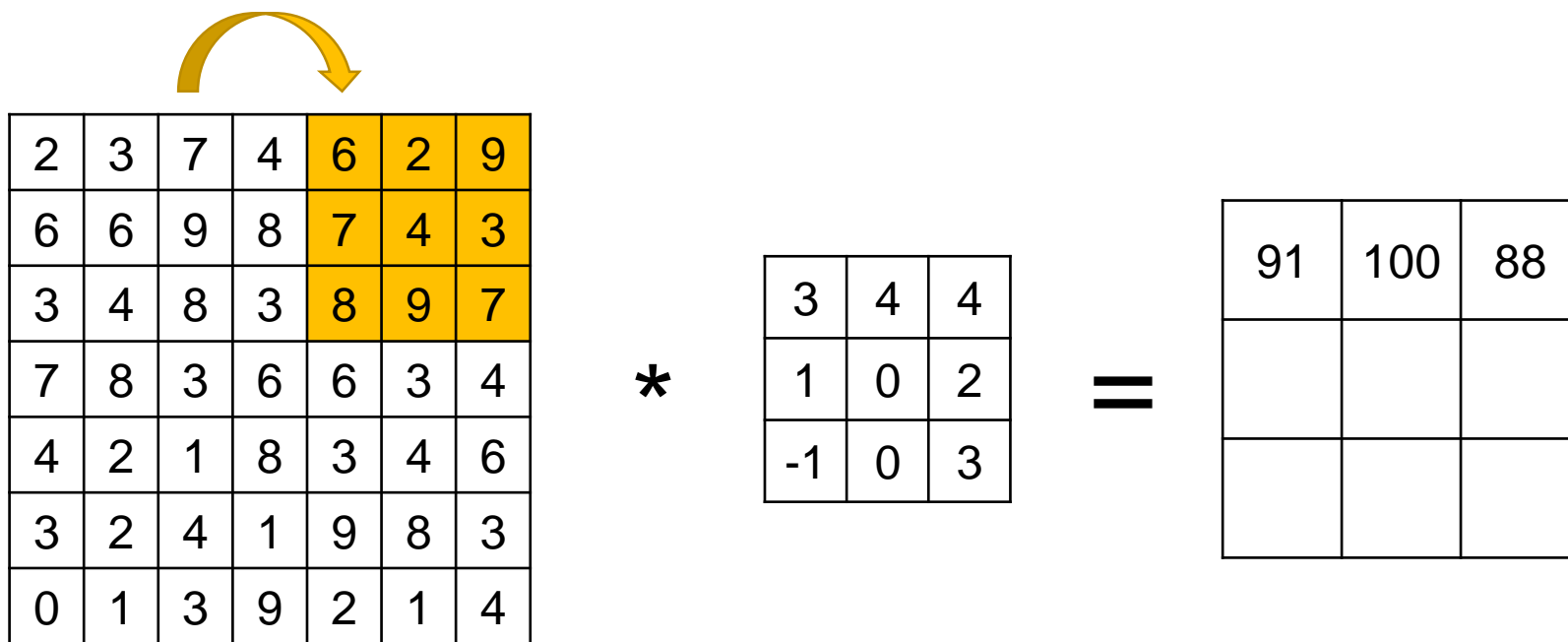
91		

# 卷积运算：步长Stride






# 卷积运算：步长Stride



# 卷积运算：步长Stride





2	3	7	4	6	2	9
6	6	9	8	7	4	3
3	4	8	3	8	9	7
7	8	3	6	6	3	4
4	2	1	8	3	4	6
3	2	4	1	9	8	3
0	1	3	9	2	1	4

\*

3	4	4
1	0	2
-1	0	3

=

91	100	88
69		

# 卷积运算：步长Stride



2	3	7	4	6	2	9
6	6	9	8	7	4	3
3	4	8	3	8	9	7
7	8	3	6	6	3	4
4	2	1	8	3	4	6
3	2	4	1	9	8	3
0	1	3	9	2	1	4

\*

3	4	4
1	0	2
-1	0	3

=

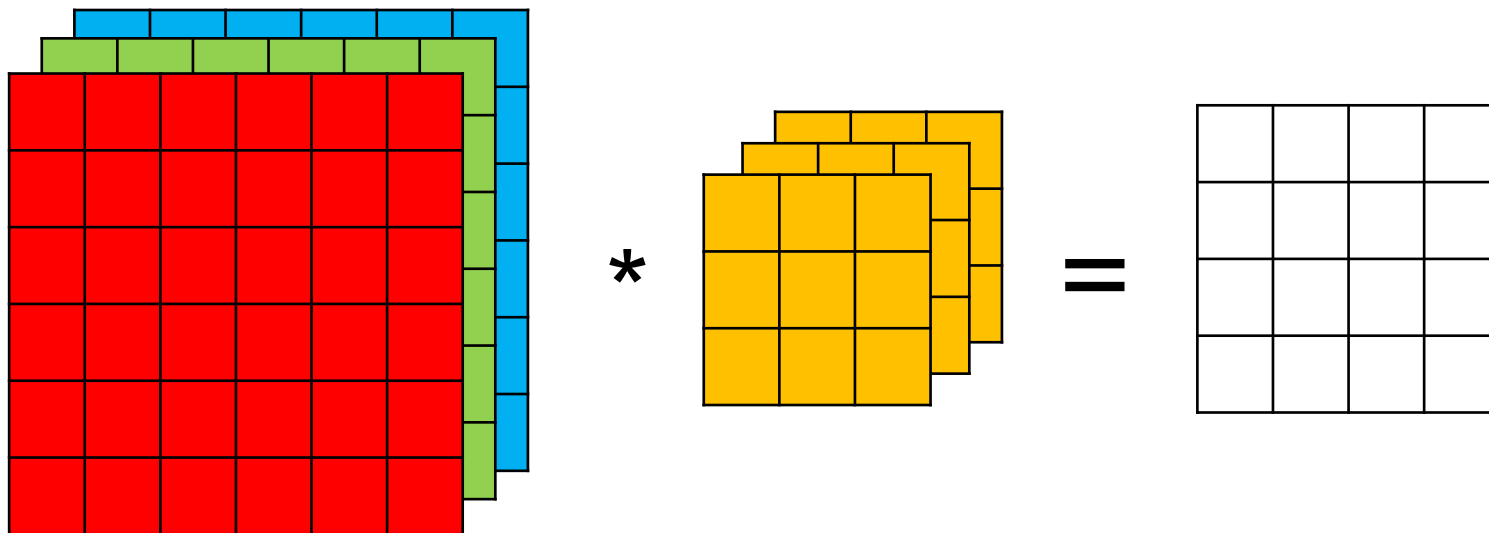
91	100	88
69	91	127
44	71	74

# 卷积运算：三维卷积



## ➤对三维立体卷积：

- RGB图像 $6 \times 6 \times 3$ ，3指的是三个颜色通道，可以想象成3个 $6 \times 6$ 图像的堆叠。
- 过滤器也是三维的，维度为 $3 \times 3 \times 3$ ，也是对应红、绿、蓝三个通道。

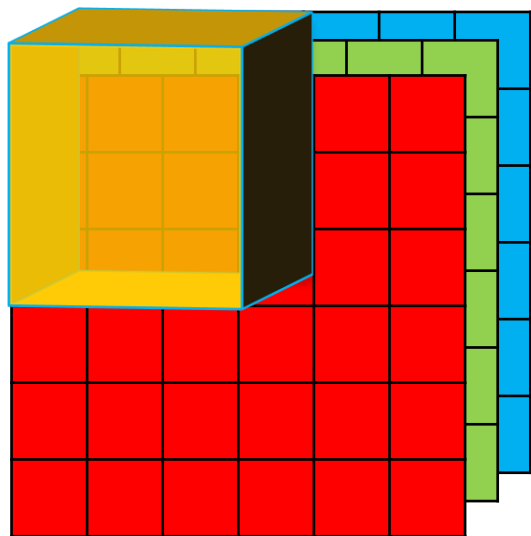


# 卷积运算：三维卷积

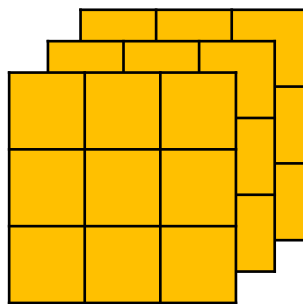


➤ 红色通道垂直边缘检测：
$$\begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix} \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

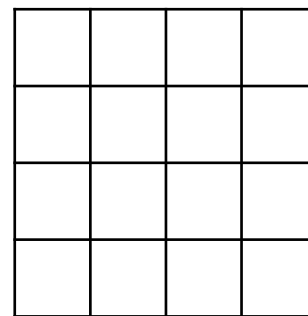
➤ 垂直边缘检测：
$$\begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}$$



\*



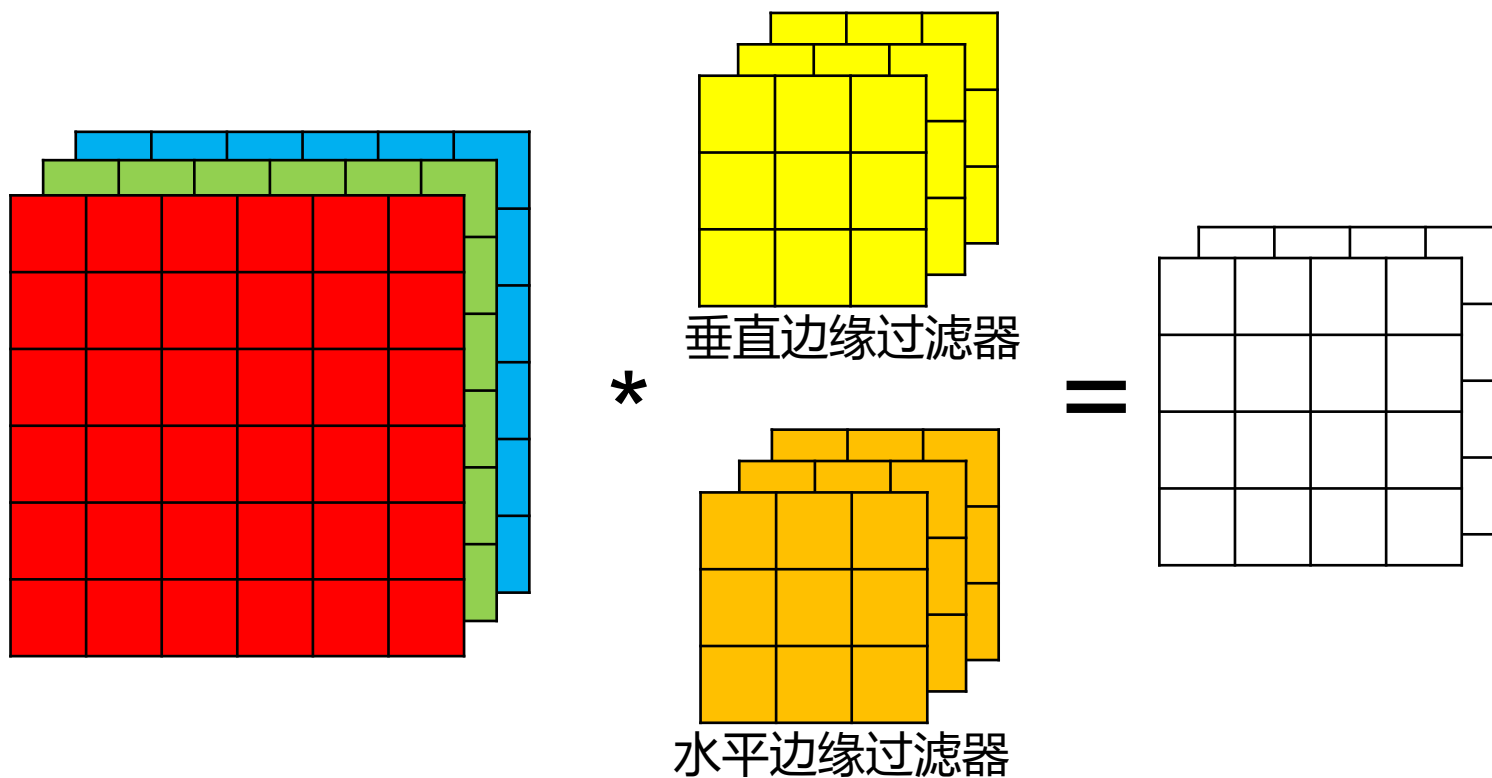
=



# 卷积运算：三维卷积



- 2个不同的特征检测过滤器分别卷积原图像，得到了2个 $4 \times 4$ 的输出。

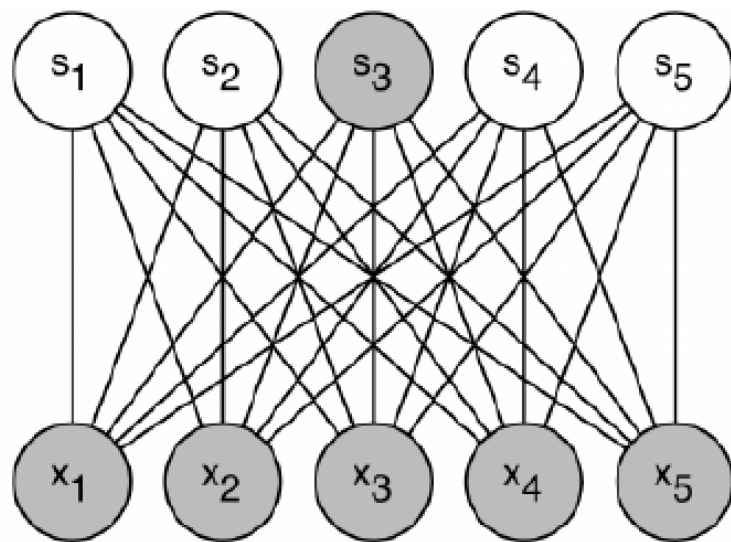
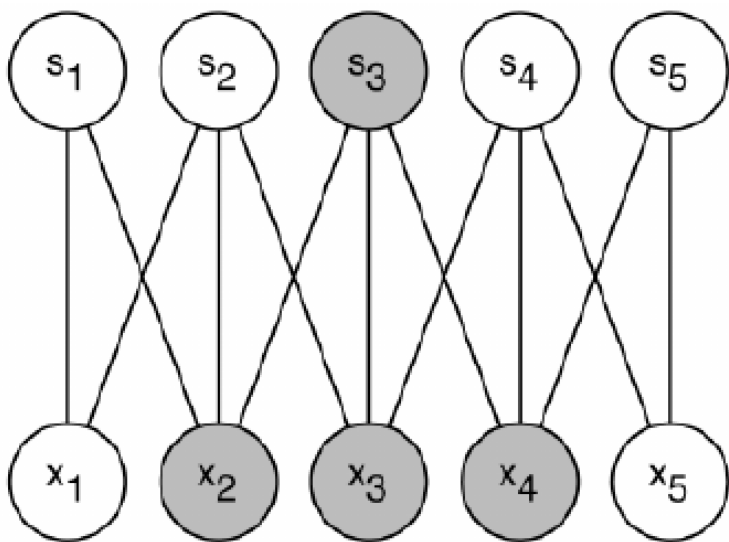


# 卷积的优势



## ➤ 稀疏连接

- 有限连接，Kernel比输入小
- 连接数少很多，学习难度小，计算复杂度低
  - $m$ 个节点与 $n$ 个节点相连 $O(mn)$
  - 限定 $k (<< m)$ 个节点与 $n$ 个节点相连，则为 $O(kn)$



# 卷积的优势



## ➤ 稀疏连接

### ➤ 有限(稀疏)连接

➤ Kernel比输入小

➤ 局部连接

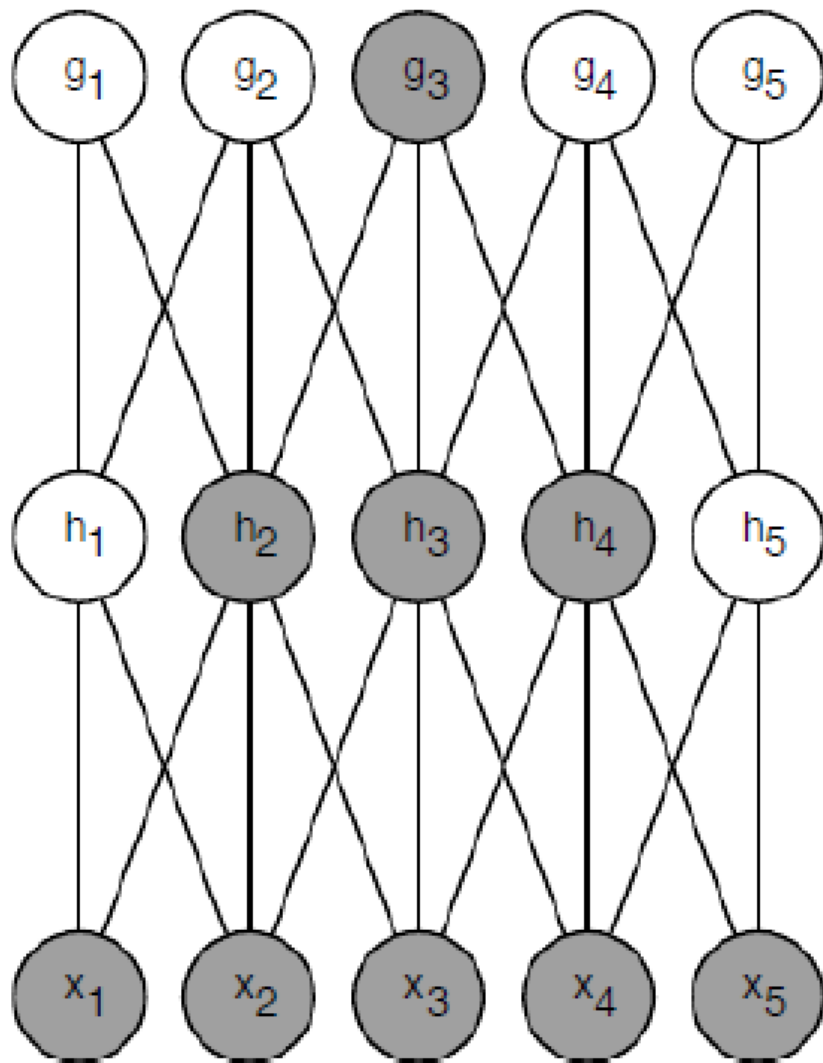
### ➤ 连接数少很多

➤ 学习难度小

➤ 计算复杂度低

### ➤ 层级感受野(Receptive Field)

➤ 越高层的神经元, 感受野越大





# 卷积的优势



## ➤ 参数共享

- 滤波器(filter)的权重不变
- 进一步极大的缩减参数数量

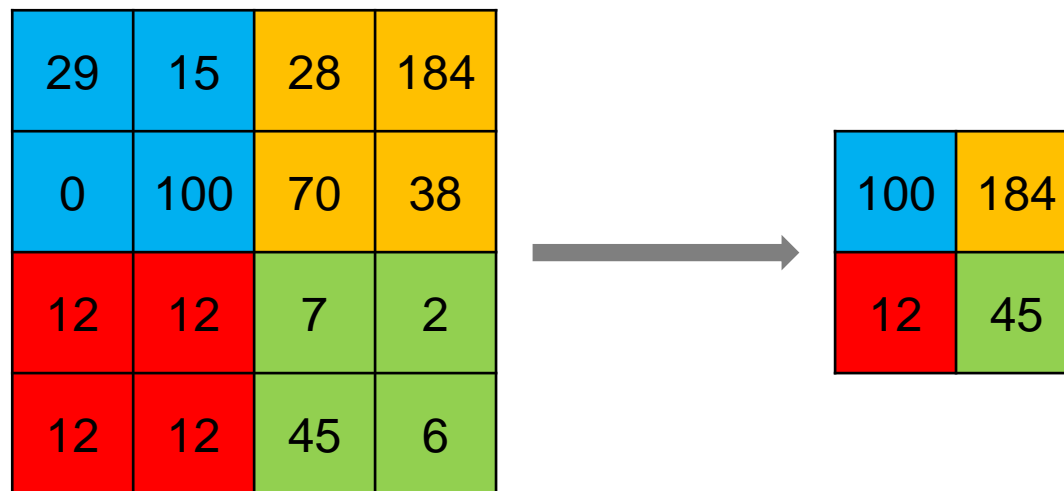
## ➤ 等变性

- 当输入改变时，输出也会以相同的方式改变

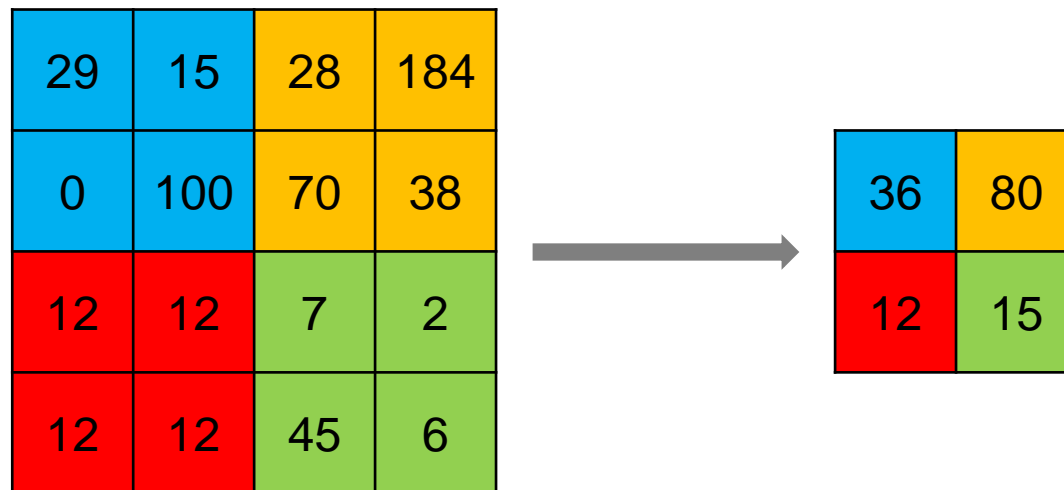
# 下采样\池化 (Pooling)



## ➤最大池化



## ➤平均池化



# 池化的作用



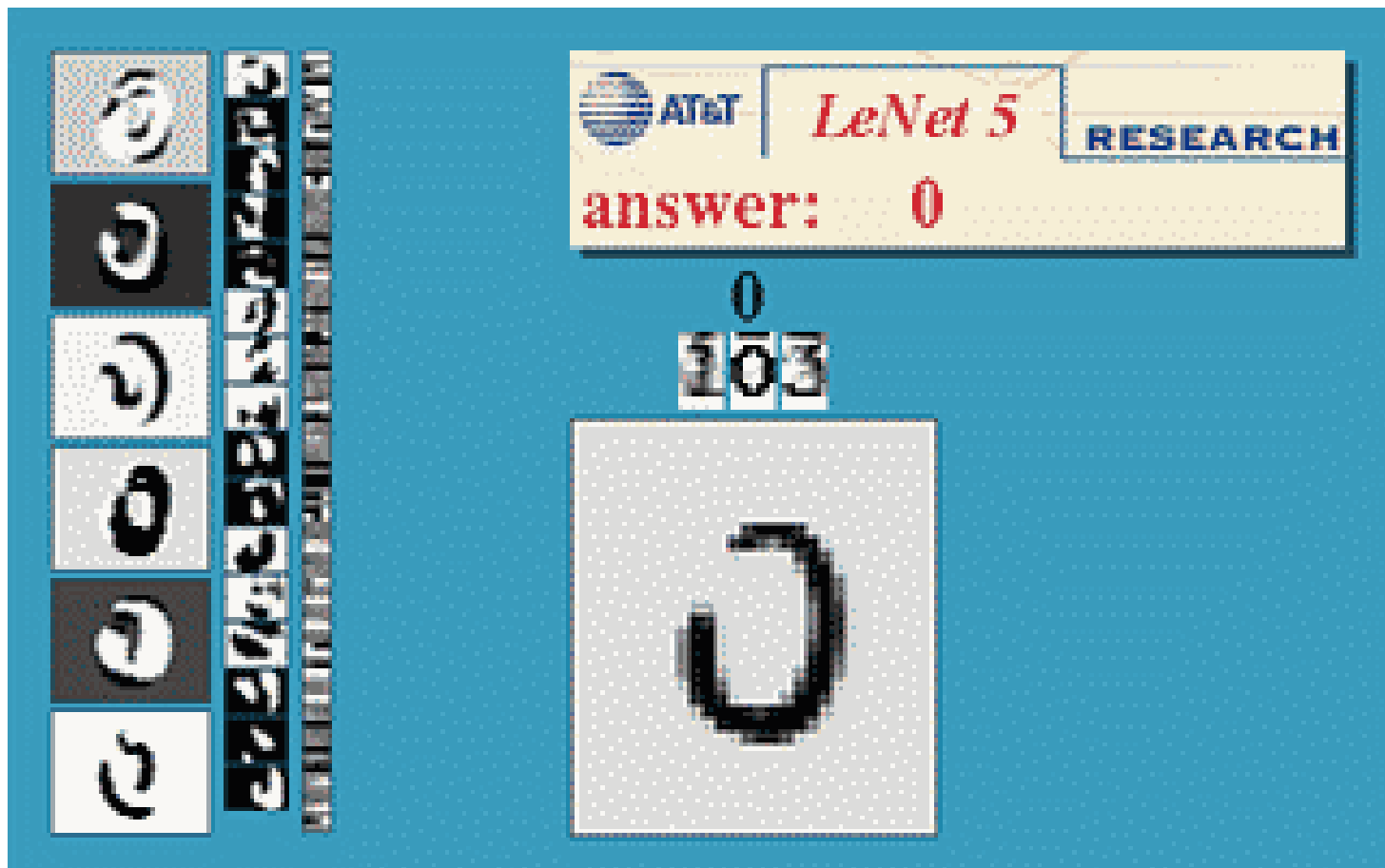
- 如果在过滤器中提取到某个特征，那么保留其最大值
- 如果没有提取到，说明可能并不存在这个特征，其中的最大值也还是很小
- 不需要学习任何值，最大池化是神经网络某一层的静态属性

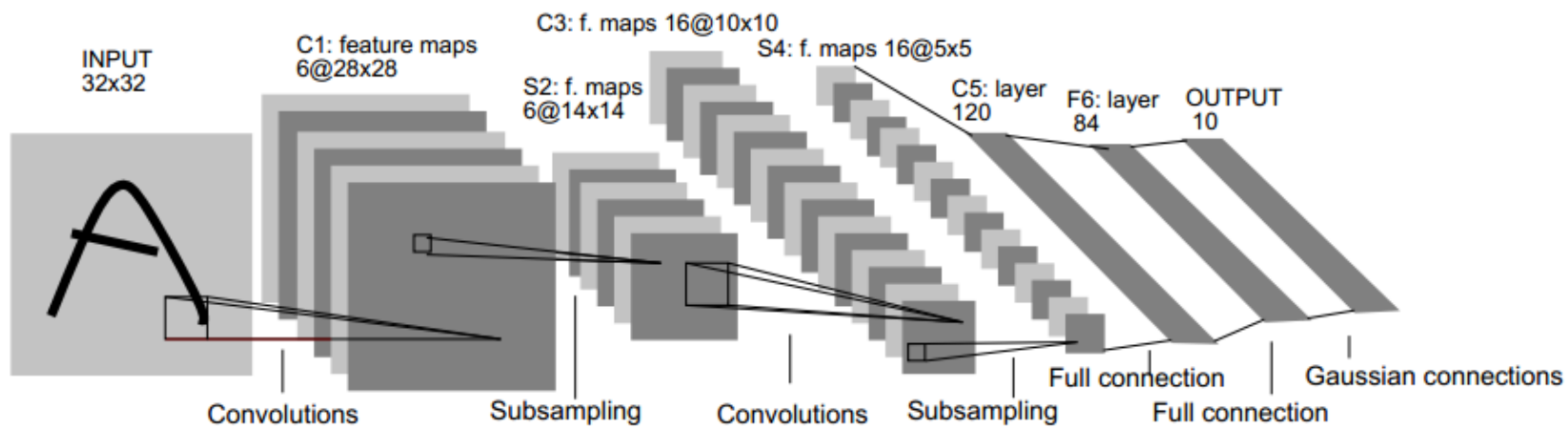


同济大学交通运输工程学院  
COLLEGE OF TRANSPORTATION ENGINEERING  
TONGJI UNIVERSITY

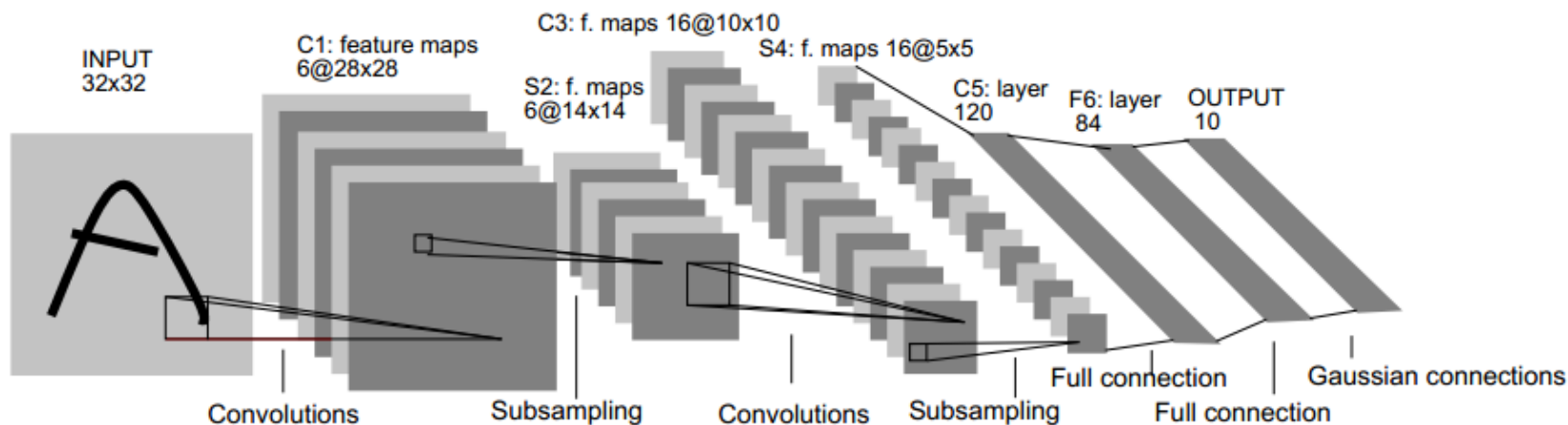
# LeNet-5手写数字识别

# LeNet-5手写数字识别

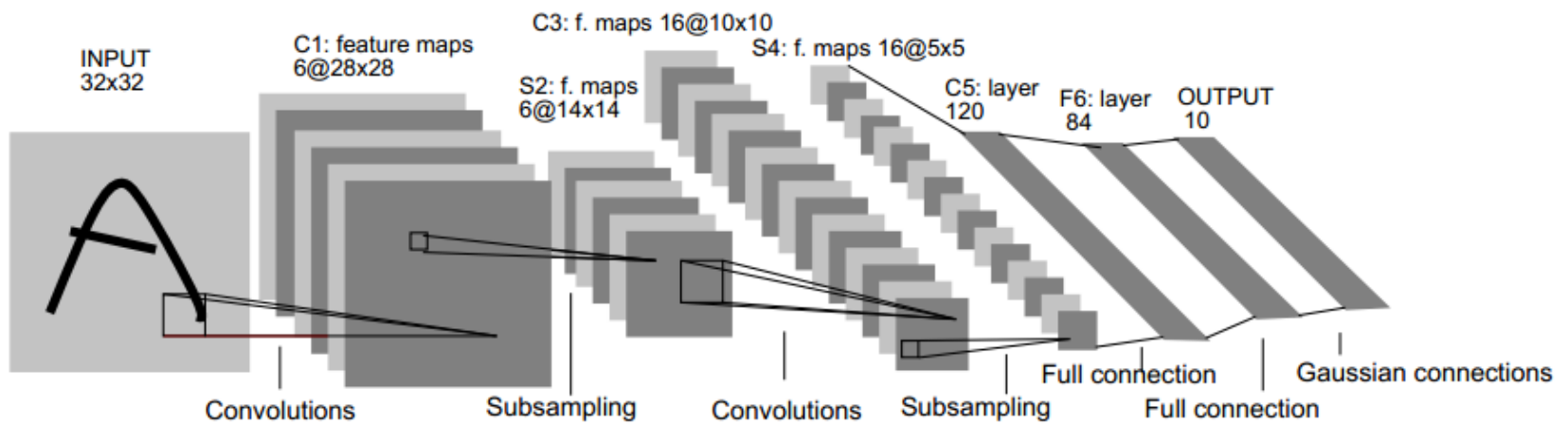




- 卷积核（卷积滤波器）
- 特征图（Feature Map）
- C层是卷积层：
  - 通过卷积运算，可以使原信号特征增强，并且降低噪音
- S层是下采样（池化）层：
  - 利用图像局部相关性的原理，对图像进行子抽样，可以减少数据处理量同时保留有用信息
- F层是经典神经网络（全连接层）：
  - 输入向量和权重向量之间的点积，再加上一个偏置。然后将其传递给sigmoid函数产生单元的一个状态。



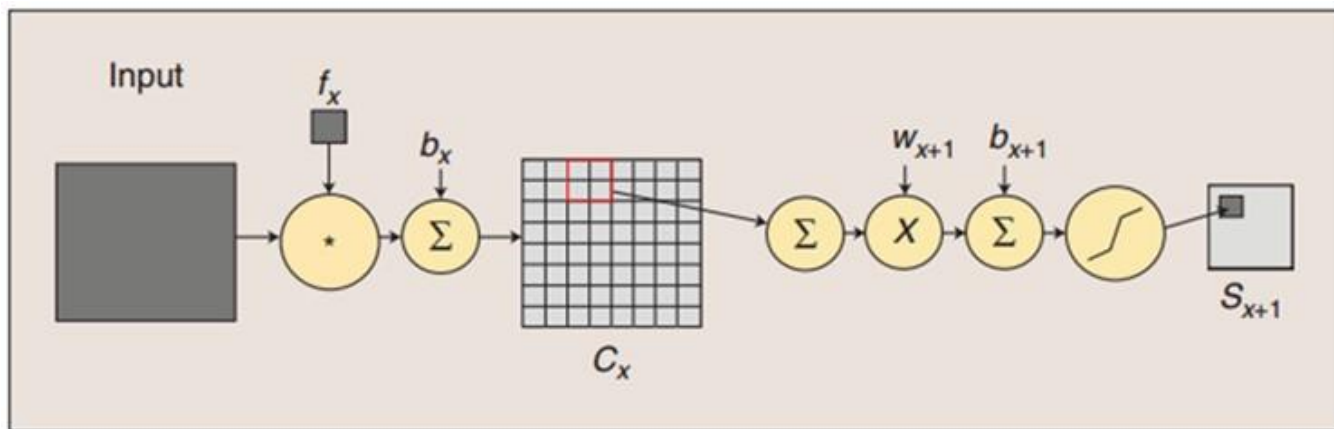
C1层		
输入图片大小	$32 \times 32$	
卷积窗大小	$5 \times 5$	
卷积窗种类	6	
输出特征图数量	6	
输出特征图大小	$28 \times 28$	$32 - 5 + 1$
神经元数量	4704	$28 \times 28 \times 6$
连接数	12304	$[(5 \times 5 + 1) \times 6] \times (28 \times 28)$
可训练参数	156	$(5 \times 5 + 1) \times 6$



S2层		
输入图片大小	$28 \times 28 \times 6$	
卷积窗大小	$2 \times 2$	
卷积窗种类	6	
输出下采样图数量	6	
输出下采样图大小	$14 \times 14 \times 6$	
神经元数量	1176	$14 \times 14 \times 6$
连接数	5880	$(2 \times 2 + 1) \times (14 \times 14) \times 6$
可训练参数	12	$6 \times (1 + 1)$



# 卷积和下采样（池化）

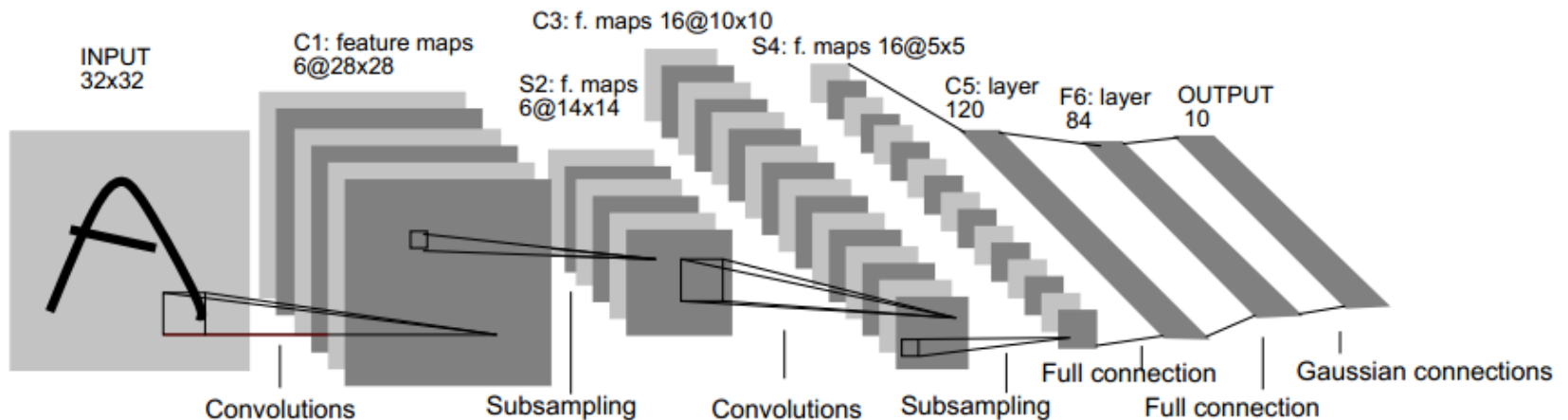


## ➤卷积

- 用一个可训练的滤波器 $f_x$ 去卷积一个输入的图片，然后加一个偏置 $b_x$ ，得到卷积层 $C_x$
- 第一阶段是输入的图片，后面的阶段就是卷积特征图

## ➤下采样（池化）

- 每邻域四个像素求和变为一个像素，然后通过标量 $w_{x+1}$ 加权，再增加偏置 $b_{x+1}$ ，然后通过一个sigmoid激活函数，产生一个大概缩小四倍的特征映射图 $S_{x+1}$



C3层		
输入图片大小	$14 \times 14 \times 6$	
卷积窗大小	$5 \times 5$	
卷积窗种类	16	
输出特征图数量	16	
输出特征图大小	$10 \times 10$	$14 - 5 + 1$
神经元数量	1600	$10 \times 10 \times 16$
连接数	151600 (部分连接)	$(60 \times 25 + 16) \times 10 \times 10$
可训练参数	1516	$60 \times 25 + 16$

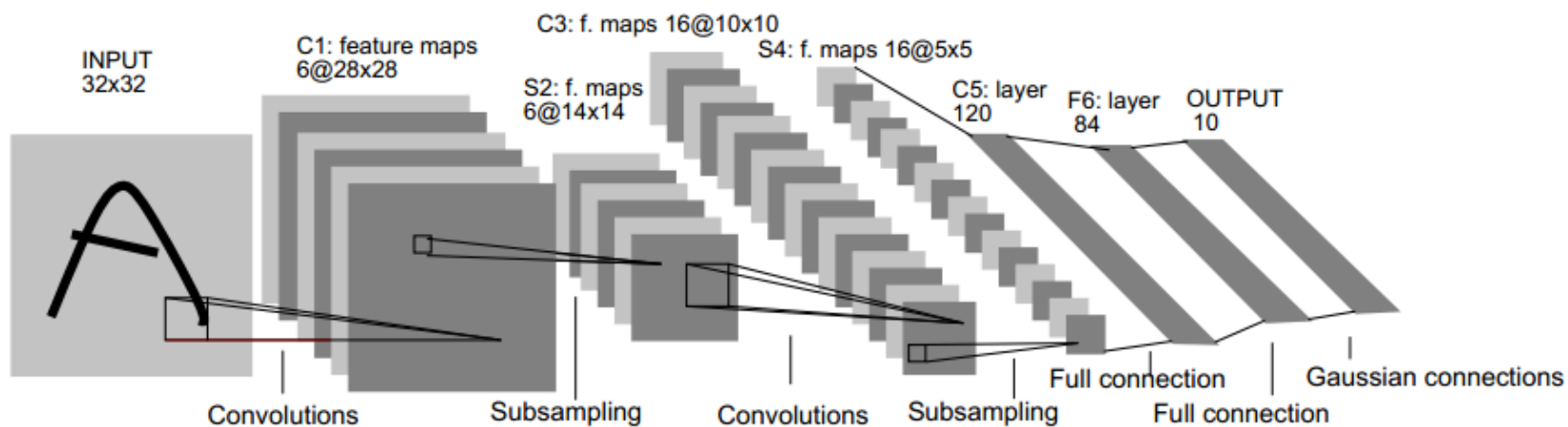
# C3层连接数



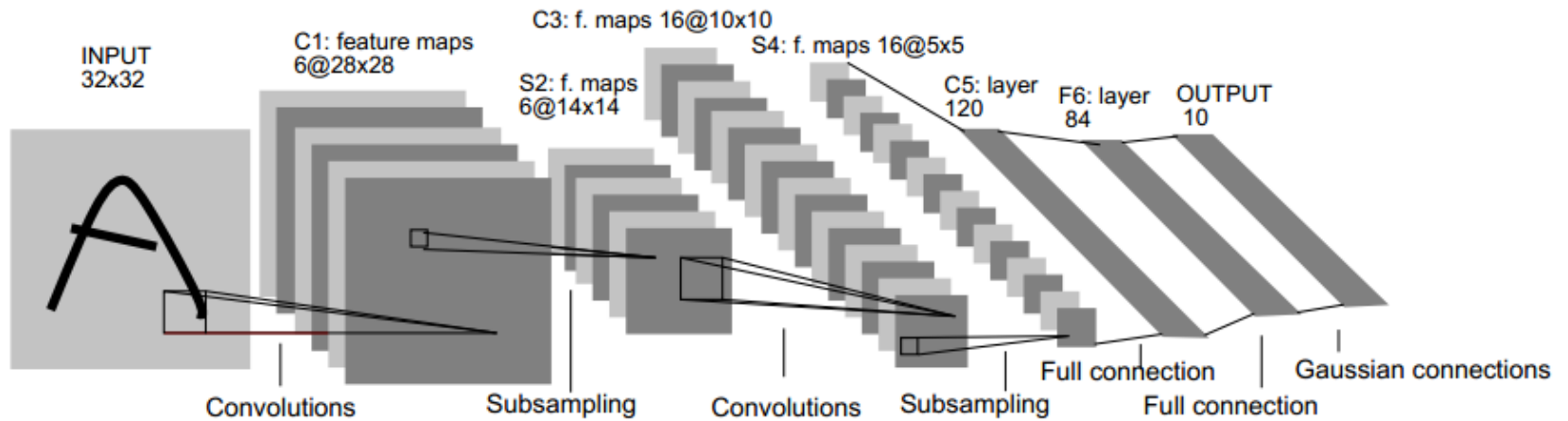
➤  $60 = 3 \times 6 + 9 \times 4 + 6$

➤  $151600 = (60 \times 25 + 16) \times 10 \times 10$

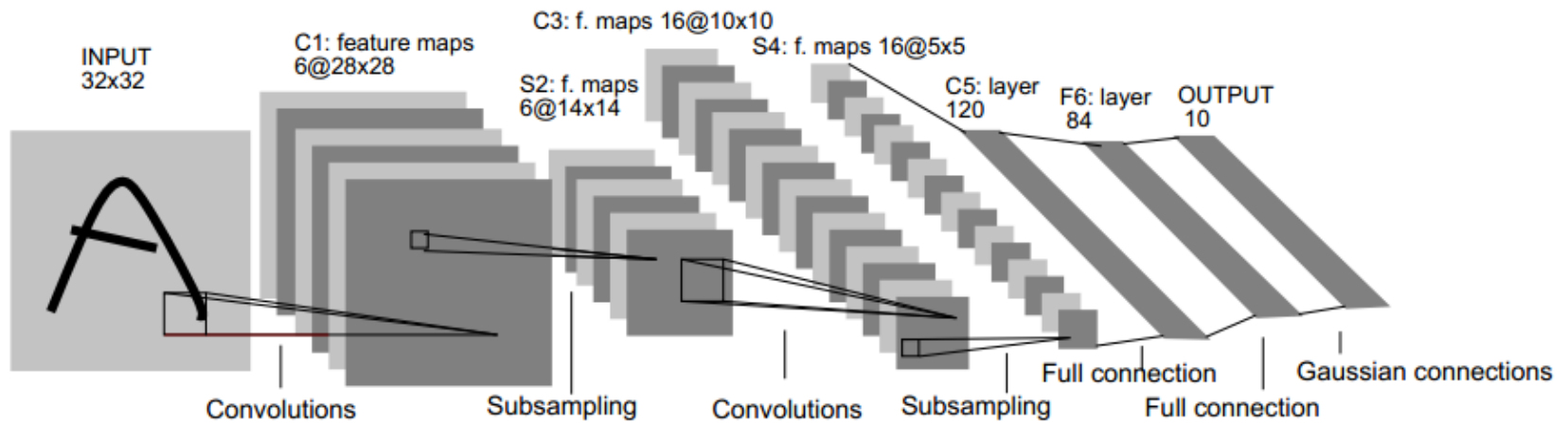
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	X				X	X	X			X	X	X	X		X	X
1	X	X				X	X	X			X	X	X	X		X
2	X	X	X				X	X	X			X		X	X	X
3		X	X	X			X	X	X	X			X		X	X
4			X	X	X			X	X	X	X		X	X		X
5				X	X	X			X	X	X	X		X	X	X



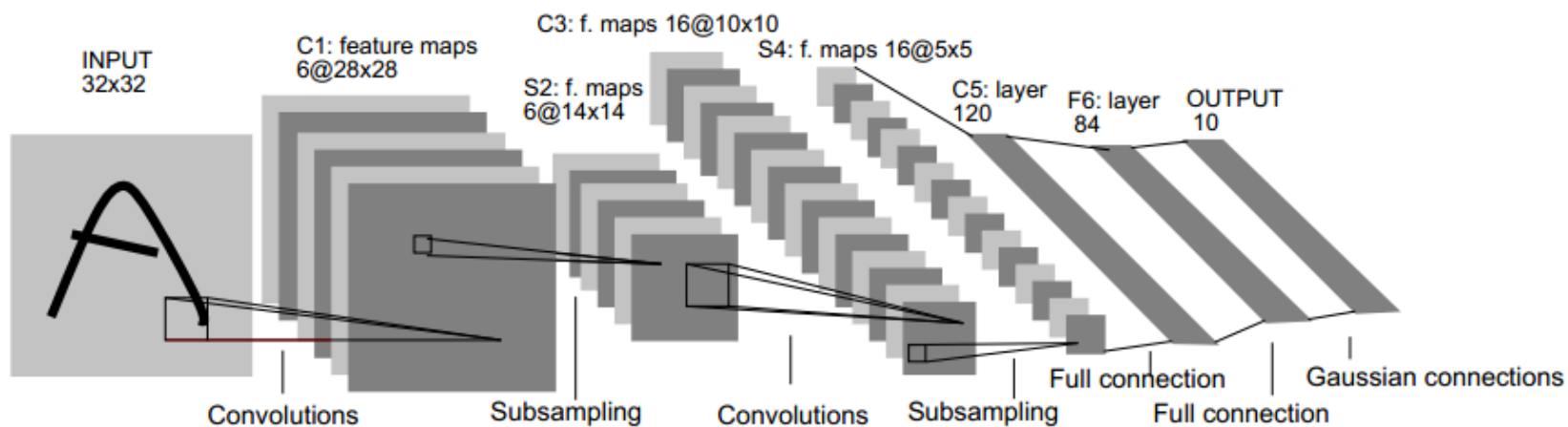
S4层		
输入图片大小	$10 \times 10 \times 16$	
卷积窗大小	$2 \times 2$	
卷积窗种类	16	
输出下采样图数量	16	
输出下采样图大小	$5 \times 5 \times 16$	
神经元数量	400	$5 \times 5 \times 16$
连接数	2000	$(2 \times 2 + 1) \times (5 \times 5) \times 16$
可训练参数	32	$16 \times (1 + 1)$



C5层		
输入图片大小	$5 \times 5 \times 16$	
卷积窗大小	$5 \times 5$	
卷积窗种类	120	
输出特征图数量	120	
输出特征图大小	$1 \times 1$	$5-5+1$
神经元数量	120	$1 \times 120$
连接数	48120 (全连接)	$(16 \times 5 \times 5 + 1) \times 1 \times 120$
可训练参数	48120	$(16 \times 5 \times 5 + 1) \times 1 \times 120$



F6层		
输入图片大小	$1 \times 1 \times 120$	
卷积窗大小	$1 \times 1$	
卷积窗种类	84	
输出特征图数量	84	
输出特征图大小	1	
神经元数量	84	
连接数	10164 (全连接)	$(120+1) \times 84$
可训练参数	10164	$(120+1) \times 84$



输出层		
输入图片大小	1×84	
输出特征图数量	10	

输出层有10个神经元，是由径向基函数单元(RBF, radial basis function)组成，输出层的每个神经元对应一个字符类别。RBF单元的输出 $y_i$ 为：

$$y_i = \sum_j (x_j - w_{ij})^2$$

# LeNet的实现：模型准备



```
import tensorflow as tf
from tensorflow.examples.tutorials.mnist import input_data
```

# 构造权重(参数)w函数, 给一些偏差(标准差=0.1)

```
def weight_variable(shape):
    initial = tf.truncated_normal(shape, stddev=0.1)
    return tf.Variable(initial)
```

# 构造偏差b函数

```
def bias_variable(shape):
    initial = tf.constant(0.1, shape=shape)
    return tf.Variable(initial)
```

# 构造卷积函数, 输入x, w与padding的形式

```
def conv2d(x, W, padding):
    return tf.nn.conv2d(x, W, strides=[1, 1, 1, 1],
                        padding=padding)
```

# 构造池化函数, kernel size=2\*2

```
def max_pool_2x2(x):
    return tf.nn.max_pool(x, ksize=[1, 2, 2, 1],
                          strides=[1, 2, 2, 1], padding='SAME')
```

步长[batch, height, width, channels]



# 输入数据与第1层卷积



# 获取MNIST数据

```
mnist = input_data.read_data_sets('MNIST_data', one_hot=True)
```

# 注册session

```
sess = tf.InteractiveSession()
```

# 定义张量输入格式

```
x = tf.placeholder(tf.float32, shape=[None, 784])
```

```
y_ = tf.placeholder(tf.float32, shape=[None, 10])
```

# 把x从[None, 784]变成[28, 28, 1], -1表示样本量不固定

```
x_image = tf.reshape(x, [-1, 28, 28, 1])
```

# 第一层卷积

```
W_conv1 = weight_variable([5, 5, 1, 6])
```

```
b_conv1 = bias_variable([6])
```

```
h_conv1 = tf.nn.relu(conv2d(x_image, W_conv1, 'SAME') + b_conv1)
```

# 第一层池化

```
h_pool1 = max_pool_2x2(h_conv1)
```

# 第2层卷积与第3、4层全连接



# 第二层卷积

```
W_conv2 = weight_variable([5, 5, 6, 16])
```

```
b_conv2 = bias_variable([16])
```

```
h_conv2 = tf.nn.relu(conv2d(h_pool1, W_conv2, 'VALID') + b_conv2)
```

# 第二层池化

```
h_pool2 = max_pool_2x2(h_conv2)
```

# 第一层全连接层, 使用ReLU激活函数

```
W_fc1 = weight_variable([5 * 5 * 16, 120])
```

```
b_fc1 = bias_variable([120])
```

# reshape改变张量结构变成一维

```
h_pool2_flat = tf.reshape(h_pool2, [-1, 5 * 5 * 16])
```

```
h_fc1 = tf.nn.relu(tf.matmul(h_pool2_flat, W_fc1) + b_fc1)
```

# 第二层全连接层, 使用ReLU激活函数

```
W_fc2 = weight_variable([120, 84])
```

```
b_fc2 = bias_variable([84])
```

```
h_fc2 = tf.nn.relu(tf.matmul(h_fc1, W_fc2) + b_fc2)
```

# 构造loss函数、梯度下降



# 输出层, 使用softmax函数

```
W_fc3 = weight_variable([84, 10])
```

```
b_fc3 = bias_variable([10])
```

```
y_conv = tf.nn.softmax(tf.matmul(h_fc2, W_fc3) + b_fc3)
```

# 构造loss函数

```
loss = -tf.reduce_sum(y_ * tf.log(y_conv))
```

# 构造accuracy函数

```
equal = tf.equal(tf.argmax(y_conv, 1), tf.argmax(y_, 1))
```

```
accuracy = tf.reduce_mean(tf.cast(equal, tf.float32))
```

# 使用梯度下降法

```
train_step = tf.train.GradientDescentOptimizer(1e-4).minimize(loss)
```

# 运行session

```
sess.run(tf.global_variables_initializer())
```

# 训练与测试模型



```
# 训练20000次, 每次batch大小为50
for i in range(20000):
    batch = mnist.train.next_batch(50)
    # 每100次训练查看训练结果
    if i % 100 == 0:
        train_accuracy = accuracy.eval(feed_dict={x: batch[0],
                                                    y_: batch[1]})
        print("step %d, train_accuracy %g" % (i, train_accuracy))
    train_step.run(feed_dict={x: batch[0], y_: batch[1]})

# 分10批测试数据, 节省显存
a = 10
b = 50
s = 0
for i in range(a):
    testSet = mnist.test.next_batch(b)
    c = accuracy.eval(feed_dict={x: testSet[0], y_: testSet[1]})
    s += c * b
print("Final test accuracy %g" % (s / (b * a)))
```

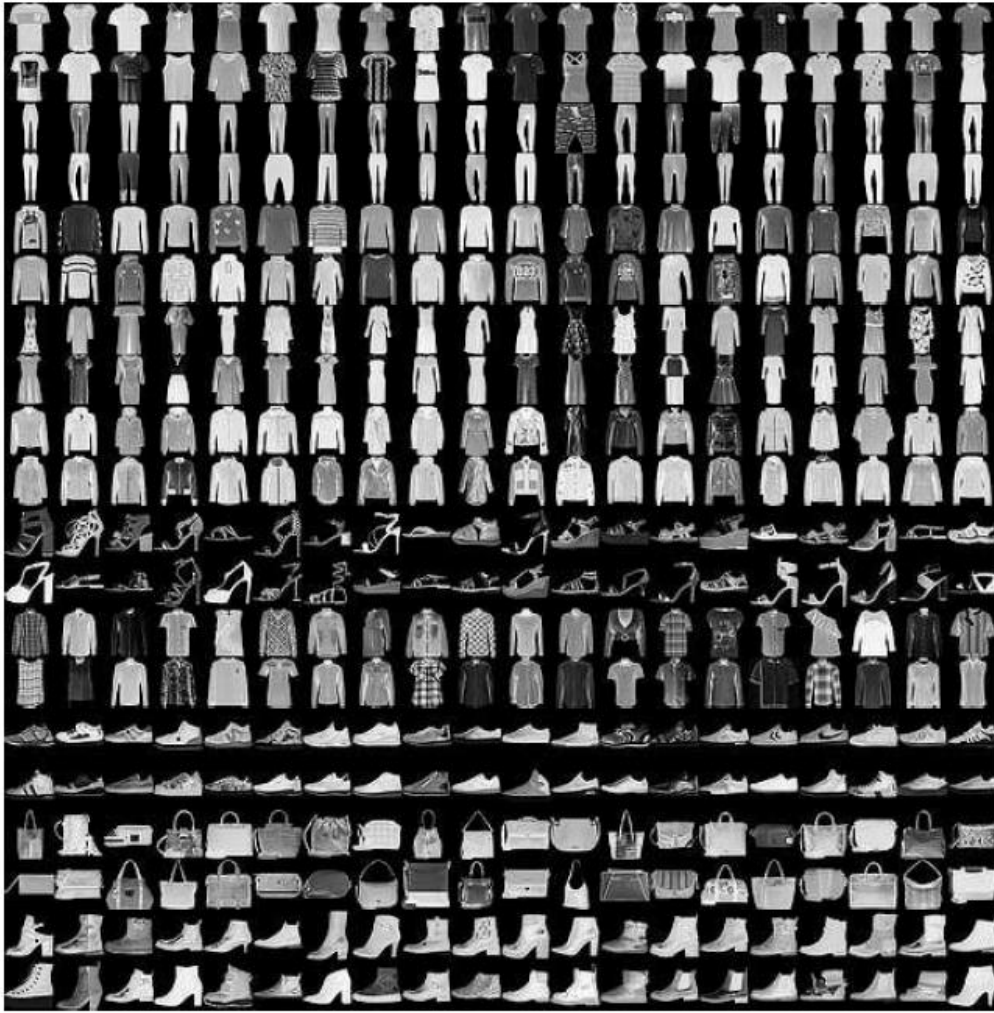
# 运行结果 (每1000次输出)



- step 0, train\_accuracy 0.1
- step 1000, train\_accuracy 0.96
- step 2000, train\_accuracy 0.92
- .....
- step 16000, train\_accuracy 0.94
- step 17000, train\_accuracy 1
- step 18000, train\_accuracy 1
- step 19000, train\_accuracy 0.98
- Final test accuracy 0.978

# Fashion MNIST



Label	Description	Examples
0	T-Shirt/Top	
1	Trouser	
2	Pullover	
3	Dress	
4	Coat	
5	Sandals	
6	Shirt	
7	Sneaker	
8	Bag	
9	Ankle boots	

# Fashion-MNIST简介



- Fashion-MNIST是一个替代MNIST手写数字集的图像数据集。
  - 由一家德国的时尚科技公司旗下的研究部门提供
  - 涵盖了来自10种类别的共7万个不同商品的正面图片
- Fashion-MNIST的大小、格式和训练集/测试集划分与原始的MNIST完全一致。
  - 60000/10000的训练测试数据划分
  - 28x28的灰度图片
- 你可以直接用它来测试你的机器学习和深度学习算法性能，且不需要改动任何的代码。

# MNIST的替代数据集



- Fashion-MNIST的目的是要成为MNIST数据集的一个直接替代品。
  - 作为算法作者，你不需要修改任何的代码，就可以直接使用这个数据集。Fashion-MNIST的图片大小，训练、测试样本数及类别数与经典MNIST完全相同
- 取代MNIST数据集的原因
  - MNIST太简单了
    - 很多深度学习算法在测试集上的准确率已经达到99.6%
  - MNIST被用烂了
  - MNIST数字识别的任务不代表现代机器学习



# LeNet的测试结果



- step 0, train accuracy 0.06
- step 1000, train accuracy 0.82
- step 2000, train accuracy 0.82
- .....
- step 16000, train accuracy 0.86
- step 17000, train accuracy 0.92
- step 18000, train accuracy 0.94
- step 19000, train accuracy 0.8
- Final test accuracy 0.862



同济大学交通运输工程学院  
COLLEGE OF TRANSPORTATION ENGINEERING  
TONGJI UNIVERSITY

# 期末大作业

源代码+实验报告+PPT汇报

# 期末大作业



➤在LeNet-5的基础上进行改进，对Fashion-MNIST数据集进行训练与测试

➤只针对LeNet-5进行调整，**不要**用GoogLeNet, VGG等重型结构

## ➤1、提交源代码

➤训练次数不大于40,000次

➤测试准确率高于基准指标：87%

➤发邮箱：[945441387@qq.com](mailto:945441387@qq.com)（吕叶婷）

➤截止时间：2022年6月10日9:59分

➤以邮件时间戳为准

➤代码命名规则：学号\_姓拼音\_名拼音.扩展名

➤例如：1953637\_Sun\_Rongjie.py

# 期末大作业



## ➤2、实验报告

- 不要封面页，首页注明标题、姓名、学号
- **不超过10页**，默认页边距、小四、1.5倍行距
- 宋体
  - 英文、数字可以是Times New Roman或其他衬线字体
- 电子版提交截止时间：2022年6月10日9:59分
  - 以邮件时间戳为准
- 发邮箱：[945441387@qq.com](mailto:945441387@qq.com)（吕叶婷）
- 报告成绩评定原则
  - 模型设计、调试的逻辑性

# 期末大作业



## ➤3、期末汇报

- PechaKucha 20×20
  - 20页PPT, 每页20秒
- 页面比例: 16比9
- PPT设置为20秒自动播放
- PPT格式要求:
  - 三段论: 问题/目标, 方法/内容, 结果/结论

## ➤期末汇报安排

- 第一组: 6月10日10:00~12:40
- 第二组: 6月17日10:00~12:40
- 具体名单通过随机数产生



同济大学交通运输工程学院  
COLLEGE OF TRANSPORTATION ENGINEERING  
TONGJI UNIVERSITY

# 第十、十一讲 结束